НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського»

ФАКУЛЬТЕТ ПРИКЛДАНОЇ МАТЕМАТИКИ

Кафедра системного програмування та спеціалізованих комп'ютерних систем

Лабораторна робота №2

З дисципліни «**Мережеві операційні системи**»

На тему: «**Синхронізація за допомогою елементарних прийомів нижнього рівня**»

Виконав: студент IV курсу

Групи КВ-73

Тарновський А.М.

КИЇВ-2020

Завдання

Банкомат приймає заявку на виплату грошей на суму X, порівнює її з кількістю грошей на рахунку клієнта Y і, якщо X<Y, здійснює спробу виплати певними купюрами номіналом 1, 2, 5, 10, 20, 50, 100 грн. Величини X і Y – випадкові. числа. Початкова кількість купюр різних номіналів у банкоматі задається студентом. Якщо сума X підлягає видачі, програма визначає, чи можна цю суму виплатити наявними у даний момент купюрами. Якщо так – гроші видаються (також коригується кількість наявних у банкоматі купюр відповідного номіналу), якщо ні – банкомат повідомляє про відмову.

Вимоги на видачу грошей надходять до банкомату після чергового обслуговування клієнта (або відмови).

Кількість терміналів і процесів: Модель автомата представити у вигляді двох взаємодіючих процесів *A і B*.

Процес *A* визначає факт надходження вимоги на виплату та можливість виплати потрібної суми, фіксує можливість виплати.

Процес *B* очікує момент появи необхідності зробити виплату і, якщо виплата можлива, визначається необхідна кількість купюр кожного номіналу для виплати, коригується рахунок клієнта і кількість готівки у банкоматі, йде виплата. Якщо виплата неможлива – клієнту надається про це повідомлення.

Засоби синхронізації: Для організації доступу до подільних ресурсів використати *алгоритм Деккера*.

Лабораторна робота виконана на мові програмування JavaScript у вигляді функціонального сайту з моделюванням роботи банкомату.

Для взаємодії використовується інтерфейс у вигляді банкомату. Для аналізу – логер.

ATM.js (main.js)

```
import React, {Fragment} from "react";
import s from './atm.module.css'
import {generateFromTo} from "./utils/random";
import {moneyParser} from "./utils/moneyValueParser";

export default class extends React.Component {
    constructor(props) {

        super(props);
        this.state = {
            atmStatus: "idle",
            pin: "",
            sum: "",
            rest: 850,
            loggerState: '',
            money: {
                one: generateFromTo(0, 50),
                two: generateFromTo(0, 50),
                five: generateFromTo(0, 50),
                ten: generateFromTo(0, 50),
                twenty: generateFromTo(0, 50),
                fifty: generateFromTo(0, 50),
                hundred: generateFromTo(0, 50),
            },
            toGive: false,
            approveGive: false,
            moneyTransferRequest: false,
            moneyTransferConfirm: false,
        }
    }

    delay = 50;
    _turn = 1;
    _trayFlag = false;
    _transferFlag = false;
    _atmTrayProcess = null;
    _transferCreatingProcess = null;

    componentDidMount() {
        window.onkeydown = (e) => {
            if (e.key === 'Escape') {
                clearInterval(this._atmTrayProcess)
                clearInterval(this._transferCreatingProcess)
            }
        }
```

```
    }

    _getUpdatedMoney = () => {
        let money = this.state.money;
        for (const key in this.state.toGive) {
            money[key] -= this.state.toGive[key]
        }

        return money
    }

    _calcAtmBalance = () => {
        let atm_balance = 0;
        for (const key in this.state.money) {
            switch (key) {
                case 'one':
                    atm_balance += this.state.money[key]
                    break;
                case 'two':
                    atm_balance += 2 * this.state.money[key]
                    break;
                case 'five' :
                    atm_balance += 5 * this.state.money[key]
                    break;
                case 'ten' :
                    atm_balance += 10 * this.state.money[key]
                    break;
                case 'twenty' :
                    atm_balance += 20 * this.state.money[key]
                    break;
                case 'fifty' :
                    atm_balance += 50 * this.state.money[key]
                    break;
                case 'hundred' :
                    atm_balance += 100 * this.state.money[key]
                    break;
                default:
                    break;
            }
        }
        return atm_balance;
    }

    _addToLogger = (str) => {
        this.setState({
            loggerState: this.state.loggerState + '\n' + str
        })
    }

    _buttonsHandler = (buttonID) => {
        if (buttonID === 0) {
            switch (this.state.atmStatus) {
                case "idle":
                    this.setState({atmStatus: "pin"});
                    this._addToLogger(`ATM GENERATED: ${JSON.stringify(this.state.money,
null, 4)}\nUSER BALANCE: ${this.state.rest}\nATM BALANCE: ${this._calcAtmBalance()}`)
                    this._atmTrayProcess = setInterval(async () => {
```

```
                this._trayFlag = true;
                while (this._transferFlag) {
                    if (this._turn) {
                        this._trayFlag = false;
                        while (this._turn) {}
                        this._trayFlag = true;
                    }
                }

                this._addToLogger('\n==============\nA-PROCESS (TRAY) GOT LOCKED ZONE');

                if (this.state.toGive === false) {
                    this._addToLogger('Заявки на виплату не знайдено. Вихід із критичної зони.\n==============')
                    this._turn = 1;
                    this._trayFlag = false;
                } else {
                    this._addToLogger('Надійшла заявка на виплату\nОбробка...')
                    if (+this.state.sum <= this.state.rest) {
                        this.setState({
                            approveGive: true
                        }, () => {
                            this._addToLogger('Запит схвалено. Вихід із критичної зони.\n==============')
                            this._turn = 1;
                            this._trayFlag = false;
                        })
                    }
                }
            }, this.delay + 10)

            this._transferCreatingProcess = setInterval(async () => {
                this._transferFlag = true;
                while (this._trayFlag) {
                    if (!this._turn) {
                        this._transferFlag = false;
                        while (!this._turn) {}
                        this._transferFlag = true;
                    }
                }
                this._addToLogger('\n==============\nB-PROCESS (TRANSFER) GOT LOCKED ZONE');

                if (this.state.approveGive === false) {
                    this._addToLogger('Запланованих виплат немає. Вихід із критичної зони\n==============')
                    this._turn = 0;
                    this._transferFlag = false;
                } else {
                    this._addToLogger('В наявності 1 запланована виплата. Обробка...')

                    const toGive = moneyParser(+this.state.sum, this.state.money);

                    this._addToLogger(toGive.enough === false
                        ? 'Неможливо виплатити сумму. Вихід із критичної
```

```
зони\n==============='
                                    : 'Є Можливість виплати. Обробка...'
                                )
                                this.setState({
                                    money: this._getUpdatedMoney(),
                                    rest: this.state.rest - +this.state.sum,
                                }, () => {
                                    this._addToLogger('Видача пройшла успішно. Вихід із
критичної зони\n==============')
                                    this.setState({
                                        sum: "",
                                        atmStatus: 'idle',
                                        toGive: false,
                                        approveGive: false,
                                    }, () => {
                                        this._turn = 0;
                                        this._transferFlag = false;
                                    })
                                })
                            }
                        }, this.delay)
                        break;
                    case "menu":
                        this.setState({atmStatus: "watch_rest"});
                        break;
                    default:
                        break;
                }
            } else if (buttonID === 1) {
                switch (this.state.atmStatus) {
                    case "menu":
                        this.setState({atmStatus: "my_sum"});
                        break;
                    default:
                        break;
                }
            } else if (buttonID === 2) {
                switch (this.state.atmStatus) {
                    case "menu":
                        this.setState({atmStatus: "pre_sum"});
                        break;
                    default:
                        break;
                }
            } else if (buttonID === 3) {
                switch (this.state.atmStatus) {
                    case "menu":
                        this.setState({atmStatus: "idle"});
                        break;
                    case "watch_rest":
                        this.setState({atmStatus: "menu"});
                        break;
                    default:
                        break;
                }
            } else if (buttonID === 4) {
                switch (this.state.atmStatus) {
```

```
                case "pre_sum":
                    break;
                default:
                    break;
            }
        } else if (buttonID === 5) {
            switch (this.state.atmStatus) {
                case "my_sum":
                    // action
                    this._addToLogger(`Запит на зняття ${this.state.sum}`)
                    this.setState({
                        toGive: true
                    })
                    break;
                case "pin":
                    this.setState({atmStatus: "menu"});
                    break;
                default:
                    break;
            }
        } else if (buttonID === 6) {
            switch (this.state.atmStatus) {
                case "my_sum":
                    this.setState({sum: "", atmStatus: "menu"});
                    break;
                case "pin":
                    this.setState({pin: "", atmStatus: "idle"});
                    break;
                default:
                    break;
            }
        } else if (buttonID === 7) {
            switch (this.state.atmStatus) {
                case "my_sum":
                    this.setState({sum: this.state.sum.substr(0, this.state.sum.length -
1)});
                    break;
                case "pin":
                    this.setState({pin: this.state.pin.substr(0, this.state.pin.length -
1)});
                    break;
                case "pre_sum":
                    this.setState({atmStatus: "menu"})
                    break;
                default:
                    break;
            }
        }
    }

    _screenContent = () => {
        switch (this.state.atmStatus) {
            case "idle":
                return (<p>{`<-- СТАРТ`}</p>)
            case "pin":
                return (
                    <Fragment>
```

```
                    <input className={s.pin} value={this.state.pin} disabled={true}/>
                    <div className={s.pinFields}>
                        <p>{`Підтв. ---->`}</p>
                        <p>{`Відм. ---->`}</p>
                        <p>{`Стерт. ---->`}</p>
                    </div>
                </Fragment>
            )
        case "menu":
            return (
                <div className={s.menuFields}>
                    <p>{`<-- баланс`}</p>
                    <p>{`<-- Зняти свою сум.`}</p>
                    <p>{`<-- Зняти задану сум.`}</p>
                    <p>{`<-- Відм.`}</p>
                </div>

            )
        case "watch_rest":
            return (
                <div className={s.restScreen}>
                    <p>Залишок: {this.state.rest} грн.</p>
                    <p>{`<-- Назад.`}</p>
                </div>
            )
        case "my_sum":
            return (
                <Fragment>
                    <input className={s.pin} value={this.state.sum} disabled={true}/>
                    <div className={s.pinFields}>
                        <p>{`Підтв. ---->`}</p>
                        <p>{`Відм. ---->`}</p>
                        <p>{`Стерт. ---->`}</p>
                    </div>
                </Fragment>

            )
        case "pre_sum":
            return (
                <div className={s.presum}>
                    <tr>
                        <td>5грн.</td>
                        <td>10грн.</td>
                    </tr>
                    <tr>
                        <td>20грн.</td>
                        <td>50грн.</td>
                    </tr>
                    <tr>
                        <td>100грн.</td>
                        <td>200грн.</td>
                    </tr>
                    <tr>
                        <td>Max.</td>
                        <td>Назад</td>
                    </tr>
                </div>
```

```
            )
        default:
            return "404 Page not found"
    }
}

_updatePin = (value) => {
    if (value.length > 1) return false;
    if (this.state.pin.length < 4) this.setState({pin: this.state.pin + value})
}

_updateSum = (value) => {
    if (value.length > 1) return false;
    if (this.state.sum.length < 9) this.setState({sum: this.state.sum + value})
}

_keyboardListener = (value) => {
    switch (this.state.atmStatus) {
        case "pin":
            this._updatePin(value);
            break;
        case "my_sum":
            this._updateSum(value);
            break;
        default:
            break;
    }
}

render() {
    return (
        <Fragment>
            <div className={s.atmWrapper}>
                <h1>ATM</h1>
                <div className={s.display}>
                    <div className={s.buttons}>
                        <button onClick={() => this._buttonsHandler(0)}/>
                        <button onClick={() => this._buttonsHandler(1)}/>
                        <button onClick={() => this._buttonsHandler(2)}/>
                        <button onClick={() => this._buttonsHandler(3)}/>
                    </div>
                    <div className={s.screen}>
                        {this._screenContent()}
                    </div>
                    <div className={s.buttons}>
                        <button onClick={() => this._buttonsHandler(4)}/>
                        <button onClick={() => this._buttonsHandler(5)}/>
                        <button onClick={() => this._buttonsHandler(6)}/>
                        <button onClick={() => this._buttonsHandler(7)}/>
                    </div>
                </div>
                <table className={s.keyboard} onClick={(e) => {
                    this._keyboardListener(e.target.textContent);
                }}>
                    <tbody>
                    <tr>
                        <td>7</td>
```

```
                            <td>8</td>
                            <td>9</td>
                        </tr>
                        <tr>
                            <td>4</td>
                            <td>5</td>
                            <td>6</td>
                        </tr>
                        <tr>
                            <td>1</td>
                            <td>2</td>
                            <td>3</td>
                        </tr>
                        <tr>
                            <td>*</td>
                            <td>0</td>
                            <td>#</td>
                        </tr>
                        </tbody>
                    </table>
                </div>
                <div className={s.atmLogger}>
                    <h1>ATM LOGGER</h1>
                    <textarea value={this.state.loggerState} disabled/>
                </div>
            </Fragment>
        );
    }
}
```

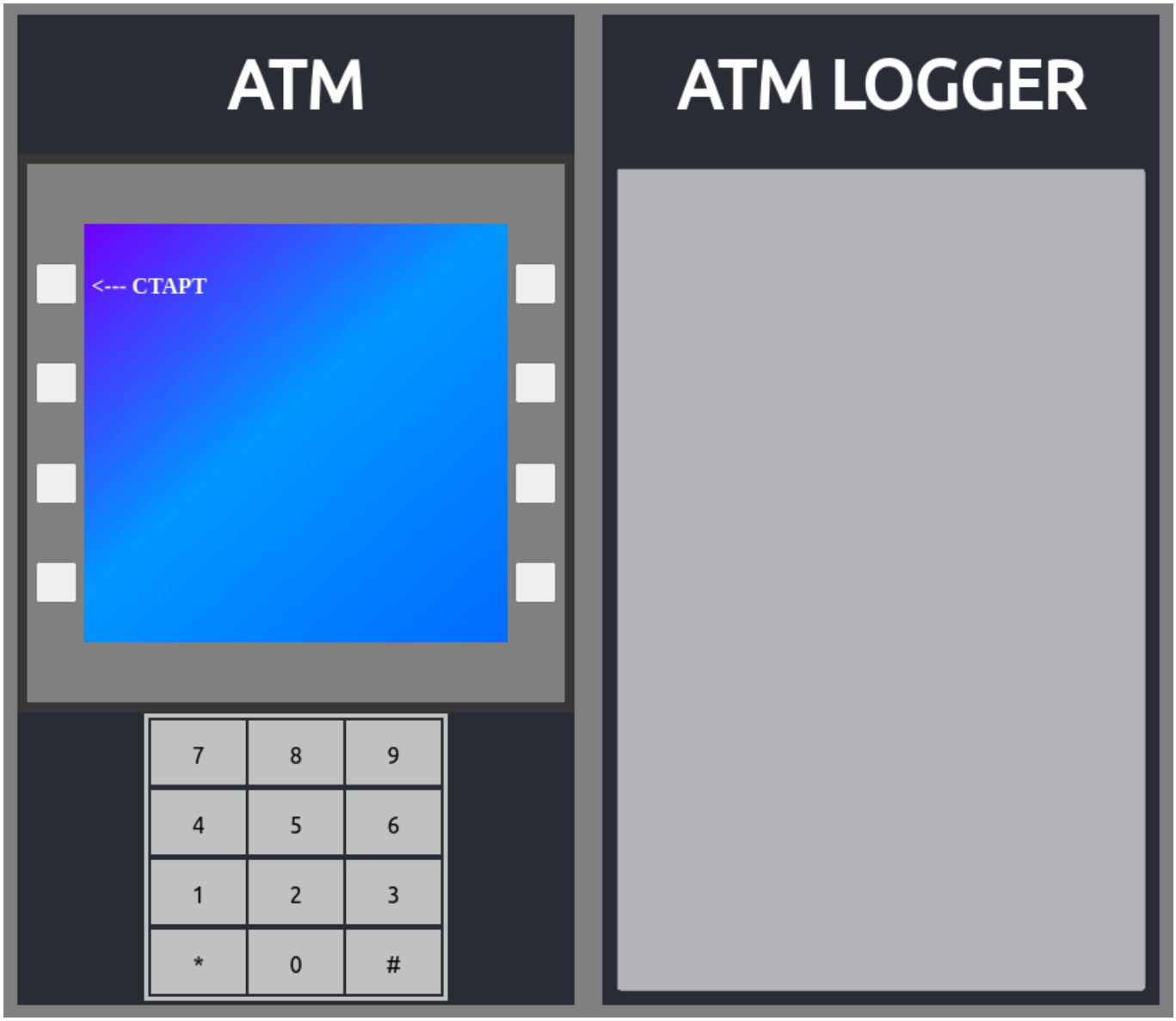
Допоміжні функції: utils/*

```
export const generateFromTo = (min = 0, max = 100) => {
    return Math.floor(min + Math.random() * (max + 1 - min))
}
const parse1 = (value, moneys) => {
    if (moneys.one > 0) {
        if (value <= moneys.one) return {one: value}
    }
    return {enough: false}
}
const parse2 = (value, moneys) => {
    if (moneys.two > 0) {
        if (getRestFromNumber(value, 2) === 0) {
            if (value / 2 <= moneys.two) return {two: value / 2}
            return Object.assign({two: moneys.two}, parse1(value - 2 * moneys.two,
moneys))
        }
        let counter = (value - getRestFromNumber(value, 2)) / 2
        if (counter <= moneys.two) {
            return Object.assign({two: counter}, parse1(value - counter * 2, moneys))
        }
    }
    return Object.assign({two: moneys.two}, parse1(value - moneys.two * 2, moneys))
}
const parse5 = (value, moneys) => {
    if (moneys.five > 0) {
        if (getRestFromNumber(value, 5) === 0) {
            if (value / 5 <= moneys.five) return {five: value / 5}
            return Object.assign({five: moneys.five}, parse2(value - 5 * moneys.five,
moneys))
        }
        let counter = (value - getRestFromNumber(value, 5)) / 5
        if (counter <= moneys.five) {
            return Object.assign({five: counter}, parse2(value - counter * 5, moneys))
        }
    }
    return Object.assign({five: moneys.five}, parse2(value - moneys.five * 5, moneys))
}
const parse10 = (value, moneys) => {
    if (moneys.ten > 0) {
        if (getRestFromNumber(value, 10) === 0) {
            if (value / 10 <= moneys.ten) return {ten: value / 10}
            return Object.assign({ten: moneys.ten}, parse5(value - 10 * moneys.ten,
moneys))
        }
        let counter = (value - getRestFromNumber(value, 10)) / 10
        if (counter <= moneys.ten) {
            return Object.assign({ten: counter}, parse5(value - counter * 10, moneys))
        }
    }
    return Object.assign({ten: moneys.ten}, parse5(value - moneys.ten * 10, moneys))
}
const parse20 = (value, moneys) => {
    if (moneys.twenty > 0) {
        if (getRestFromNumber(value, 20) === 0) {
            if (value / 20 <= moneys.twenty) return {twenty: value / 20}
```

```
            return Object.assign({twenty: moneys.twenty}, parse10(value - 20 * moneys.twenty, moneys))
        }
        let counter = (value - getRestFromNumber(value, 20)) / 20
        if (counter <= moneys.twenty) {
            return Object.assign({twenty: counter}, parse10(value - counter * 20, moneys))
        }
    }
    return Object.assign({twenty: moneys.twenty}, parse10(value - moneys.twenty * 20, moneys))
}
const parse50 = (value, moneys) => {
    if (moneys.fifty > 0) {
        if (getRestFromNumber(value, 50) === 0) {
            if (value / 50 <= moneys.fifty) return {fifty: value / 50}
            return Object.assign({fifty: moneys.fifty}, parse20(value - 50 * moneys.fifty, moneys))
        }
        let counter = (value - getRestFromNumber(value, 50)) / 50
        if (counter <= moneys.fifty) {
            return Object.assign({fifty: counter}, parse20(value - counter * 50, moneys))
        }
    }
    return Object.assign({fifty: moneys.fifty}, parse20(value - moneys.fifty * 50, moneys))
}
const parse100 = (value, moneys) => {
    if (moneys.hundred > 0) {
        if (getRestFromNumber(value, 100) === 0) {
            if (value / 100 <= moneys.hundred) return {hundred: value / 100}
            return Object.assign({hundred: moneys.hundred}, parse50(value - 100 * moneys.hundred, moneys))
        }
        let counter = (value - getRestFromNumber(value, 100)) / 100
        if (counter <= moneys.hundred) {
            return Object.assign({hundred: counter}, parse50(value - counter * 100, moneys))
        }
    }
    return Object.assign({hundred: moneys.hundred}, parse50(value - moneys.hundred * 100, moneys))
}

const getRestFromNumber = (dividend, divisor) => dividend % divisor

export const moneyParser = (value, moneys) => {
    switch(`${value}`.length) {
        case 1:
            return parse5(value, moneys);
        case 2:
            return parse50(value, moneys);
        default:
            return parse100(value, moneys);
    }
}
```



Вигляд програми:

Демонстрація логгера: