

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

О. І. Марченко, О. О. Марченко

ПАРАЛЕЛЬНЕ ПРОГРАМУВАННЯ ЛАБОРАТОРНИЙ ПРАКТИКУМ

*Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського
як навчальний посібник
для здобувачів ступеня бакалавра за спеціальністю
123 Комп'ютерна інженерія*

Київ
КПІ ім. Ігоря Сікорського
2021

Рецензент *Заболотня Т.М.*, канд. техн. наук, доцент, доцент, КПІ ім. Ігоря Сікорського

Відповідальний редактор *Орлова М.М.*, канд. техн. наук, доцент, доцент

*Гриф надано Методичною радою КПІ ім. Ігоря Сікорського (протокол № 2 від 09.12.2021 р.)
за поданням Вченої ради Факультету прикладної математики (протокол № 2 від 27.09.2021 р.)*

Електронне мережне навчальне видання

Марченко Олександр Іванович, канд. техн. наук, доцент
Марченко Олексій Олександрович, асистент

ПАРАЛЕЛЬНЕ ПРОГРАМУВАННЯ ЛАБОРАТОРНИЙ ПРАКТИКУМ

Паралельне програмування: лабораторний практикум з дисципліни «Паралельне програмування»: [Електронний ресурс] : навч. посіб. для студ. спеціальності 123 – «Комп’ютерна інженерія» / О. І. Марченко, О. О. Марченко ; КПІ ім. Ігоря Сікорського. – Електронні текстові дані (1 файл: 3,46 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2021. – 80 с.

Навчальний посібник розроблено для ознайомлення студентів із завданнями до лабораторних робіт з дисципліни «Паралельне програмування». Навчальний посібник містить інформацію до виконання проектів програм лабораторних робіт, яка включає постановку завдання для кожної лабораторної роботи, вимоги до проектів програм, що повинні розробити студенти, вказівки до оформлення звіту та тестування розроблених алгоритмів і відповідних їм програм, варіанти індивідуальних завдань, а також наведені контрольні питання для підготовки до захисту лабораторних робіт.

Навчальний посібник призначений для студентів очної форми навчання за спеціальністю 123 – «Комп’ютерна інженерія» факультету прикладної математики КПІ ім. Ігоря Сікорського.

© О.І. Марченко, О.О. Марченко, 2012-2021

© КПІ ім. Ігоря Сікорського, 2021

ЗМІСТ

ВСТУП	1
ЛАБОРАТОРНА РОБОТА №1. РОБОТА З КОМПІЛЯТОРАМИ МОВ С ТА JAVA В РЕЖИМІ КОМАНДНОГО РЯДКА.....	2
ЛАБОРАТОРНА РОБОТА №2. СТВОРЕННЯ ПАРАЛЕЛЬНИХ ПОТОКІВ ОПЕРАЦІЙНОЇ СИСТЕМИ LINUX ТА НАЙПРОСТІША СИНХРОНІЗАЦІЯ.....	20
ЛАБОРАТОРНА РОБОТА №3. ЗАСОБИ ВЗАЄМОДІЇ ПАРАЛЕЛЬНИХ ПОТОКІВ ОПЕРАЦІЙНОЇ СИСТЕМИ LINUX.....	24
ЛАБОРАТОРНА РОБОТА №4. КОМПЛЕКСНЕ ВИКОРИСТАННЯ ЗАСОБІВ ВЗАЄМОДІЇ ПАРАЛЕЛЬНИХ ПОТОКІВ ОПЕРАЦІЙНОЇ СИСТЕМИ LINUX.....	26
ЛАБОРАТОРНА РОБОТА №5. СТВОРЕННЯ ПАРАЛЕЛЬНИХ ПОТОКІВ МОВИ JAVA ТА ЗАСОБИ ЇХ ВЗАЄМОДІЇ	43
ЛАБОРАТОРНА РОБОТА №6. КОМПЛЕКСНЕ ВИКОРИСТАННЯ ЗАСОБІВ ВЗАЄМОДІЇ ПАРАЛЕЛЬНИХ ПОТОКІВ МОВИ JAVA	45
ІНФОРМАЦІЙНІ РЕСУРСИ	79
СПИСОК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ	79

ВСТУП

Дисципліна «Паралельне програмування» є спеціальною нормативною дисципліною у загальній схемі алгоритмічної і програмістської підготовки фахівців відповідно до освітньо-професійної програми підготовки бакалаврів за спеціальністю 123 – «Комп’ютерна інженерія» і складається з одного кредитного модуля. Вивченню дисципліни повинні передувати дисципліни "Структури даних та алгоритми", "Програмування" та "Системне програмування". Дисципліна «Паралельне програмування» забезпечує вивчення дисципліни „Архітектура комп’ютерів”.

Дисципліна «Паралельне програмування» призначена для вивчення принципів та концепцій паралельних та розподілених обчислень, а також реалізації цих принципів у конкретних операційних системах, мовах програмування та спеціалізованих бібліотеках програм.

Цикл лабораторних робіт складається з шести робіт і призначений для покриття практичної частини дисципліни «Паралельне програмування».

Роботи дозволяють отримати практичний досвід створення паралельних програм з використанням різних примітивів і механізмів синхронізації та комунікації паралельних процесів і потоків.

Інструкції та завдання включають для кожної роботи:

- постановку завдання та вимоги до алгоритмів та програм, що повинні розробити студенти;
- вказівки до оформлення звіту та тестування розробленого алгоритму і відповідної йому програми;
- контрольні питання для самоконтролю та підготовки до захисту лабораторної роботи;
- варіанти індивідуальних завдань.

ЛАБОРАТОРНА РОБОТА №1. РОБОТА З КОМПІЛЯТОРАМИ МОВ С ТА JAVA В РЕЖИМІ КОМАНДНОГО РЯДКА

Основи роботи з gcc

Основним компілятором для ОС Linux є gcc (gnu c compiler). Він входить до колекції компіляторів GCC (GNU Compilers Collection). Для перевірки чи встановлений компілятор необхідно виконати наступну команду:

```
$ gcc
```

Якщо у відповідь буде відображено:

```
gcc: no input files
```

то компілятор встановлений і готовий до роботи.

Основні опції gcc:

-OX — де X задає рівень оптимізації компілятора від 0 до 3. Чим більше число тим складніші оптимізації використовуються. При налагодженні програми необхідно використовувати рівень 0. Після того як програма була налагоджена рекомендується використовувати рівень 2. Винятком є значення s (-Os), яка вказує що необхідно оптимізувати за розміром. Така оптимізація може використовуватися для вбудованих систем у який бракує пам'яті. За замовченням використовується рівень 2;

-o <file_name> — ім'я файлу що буде отримано після компіляції (програми, об'єктного файлу чи бібліотеки);

-c — вказує на те що компілюється об'єктний файл;

-g — вказує що необхідно компілювати програму із інформацією для налагодження.

Компіляція та запуск створеної програми

Найпростіший випадок компіляції може бути представлено наступним чином:

```
gcc -o <ім'я файлу результату> <опції компіляції> <список вихідних та об'єктних файлів для компіляції>
```

Збережіть наступний фрагмент вихідного коду у файлі "hello.c". У кінці файлу рекомендується ставити порожній рядок.

```
#include <stdio.h>
int main()
{
    printf("Hello, world!\n");
    return 0;
}
```

Компілювати цю програму можна, наприклад, так:

```
$ gcc -o hello hello.c
```

Для виконання програми **hello** потрібно ввести команду:

```
$ ./hello
```

Компіляція складних програм мовою C і використання make

Утиліта **make** використовується для автоматизації процесу компіляції програми. За допомогою спеціальної командної мови можна прописати набір правил для компіляції програми. Усі правила повинні бути збережені у файлі з назвою Makefile.

Правило має наступну структуру:

```
<ціль> : <залежності>
```

```
<tab><команда 1>
```

```
<tab><команда 2>
```

```
...
```

```
<tab><команда N>
```

<ціль> — ім'я файлу, що повинно бути отримано в результаті роботи команд. Винятком є вказування замість імені файлу назви певної дії, яка буде виконана у результаті роботи команд;

<залежності> — імена файлів, що впливають на роботу команд, та зміна яких повинна призводити до повторного виконання правила (регенерації цільового файлу);

<команда X> — це команда яку необхідно виконати. Команди виконуються згори вниз. Рядки, в яких записано команди повинні починатися з символу табуляції.

Для того, щоби правило виконалося, необхідно виконання умов:

- файл цілі не існує;

- якщо файл цілі існує, але хоча б один із файлів залежностей був модифікований після створення файлу цілі.

Якщо правило є сервісним, і не передбачає створення файлу цілі, назву його цілі необхідно додати до директиви `.PHONY`, що може бути розташована на початку файлу, наприклад:

```
.PHONY: clean check config
```

Наведемо повний приклад make-файлу.

```
.PHONY: clean check config
```

Наведемо повний приклад make-файлу.

```
.PHONY: greet build rebuild run clean
```

```
greet:
```

```
@echo "Terminating make - please specify target explicitly"  
@echo "  build  : fast rebuild / build"  
@echo "  rebuild : full rebuild"  
@echo "  run    : run after fast rebuild / build"  
@echo "  clean  : perform full clean"
```

```
build: main
```

```
rebuild: clean main
```

```
run: build
```

```
./main
```

```
clean:
```

```
rm -rvf *.o main
```

```
main: main.o func.o
```

```
gcc -o main main.o func.o
```

```
main.o: main.c func.h
```

```
gcc -c -o main.o main.c
```

```
func.o: func.c func.h
```

```
gcc -c -o func.o func.c
```

Наприклад, для того, щоб почати виконання make-файлу з цілі **build**, потрібно ввести у командному рядку команду

```
$ make build
```

а для того, щоб почати виконання make-файлу з цілі **run**, потрібно ввести у командному рядку команду

```
$ make run
```

тощо.

Важливо, що директорія, в якій знаходиться Makefile з вказаним текстом, повинна бути поточною.

Обробка списку аргументів, що передаються програмі мовою C (argc, argv[])

Для того, щоб передати компілятору інформацію, що дана програма при запуску на виконання може отримувати параметри (аналогічно до того, як отримують параметри функції), при написанні головної функції main цієї програми необхідно у її заголовку описати два спеціальних аргументи (параметри) argc та argv наступним чином:

```
int main (int argc, char** argv)
```

або використати еквівалентний опис

```
int main (int argc, char* argv[])
```

де **argc** – довжина (кількість елементів) вектора argv;
argv[] – вектор вказівників на рядки, які є аргументами, що передаються у програму.

Приклад. Написати програму, яка приймає довільну кількість рядкових аргументів, розділених пробілами, і виводить їх на екран монітора.

```
#include <stdio.h>  
#include <stdlib.h>  
int main (int argc, char** argv)  
// або int main(int argc, char* argv[])  
{  
    int count;  
    for (count=1; count < argc; count++)  
        printf(" %s", argv[count]);  
    printf("\n");  
    return 0;  
}
```


Найбільш характерним використанням аргументів головної функції є передавання їй параметрів у вигляді опцій команд ОС Linux, наприклад **-l**, **--author**.

Команди ОС Linux можуть мати опції двох видів, короткі опції та довгі опції:

- коротка опція має вигляд “-x”, де x – один символ, що символізує цю опцію, наприклад “-o”;
- довга опція має вигляд “--opt_name”, де **opt_name** – назва довгої (багатосимвольної) опції, наприклад “--output”.

Для зручної обробки списку аргументів програми, які є такими опціями, у стандартній бібліотеці glibc (GNU libc) існує спеціальна функція:

```
getopt_long(int argc,  
            char* const argv,  
            const char* shortopts,  
            const struct option* longopts,  
            int* indexptr)
```

Ця функція декодує опції із вектору argv, повна довжина якого знаходиться у параметрі **argc**.

shortopts – описує список коротких опцій, які дозволені у програмі. Цей параметр є рядком символів, кожен з яких представляє собою ім'я короткої опції.

longopts – описує список довгих опцій, які дозволені у програмі.

Коли функція **getopt_long** знаходить коротку опцію, вона повертає символний код короткої опції і, якщо у неї є аргумент, то заносить його у спеціальну змінну **optarg**.

Коли функція **getopt_long** знаходить довгу опцію, вона виконує дію базуючись на полях **flag** та **val**, що описані для цієї опції в структурі типу **option**.

Структура типу **option** має наступні поля:

- 1) **const char* name** – поле представляє ім'я опції;
- 2) **int has_arg** – поле вказує, чи очікує ця опція на аргументи і може приймати значення:

- **no_argument**
- **required_argument**
- **optional_argument**

3) **int* flag**

4) **int val**

Два останніх поля **flag** та **val** вказують, як реагувати на появу даної опції:

- якщо вказівник **flag** має нульове значення, тоді функція повертає значення **val**;
- якщо **flag** є вказівником на змінну, то значення **val** присвоюється змінній, на яку вказує **flag**.

Розглянемо приклад програми, яка має один аргумент і якій у якості аргументу може передаватись або коротка опція **-t**, або довга опція **--test**.

```
#include <stdio.h>
```

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

```
#include <getopt.h>
```

```
int main(int argc, char** argv)
```

```
{    int c = 0;
```

```
    static int verbose_flag = 0;
```

```
    while(1)
```

```
    {    static struct option long_options[] =
```

```
        {"test", required_argument, 0, 't'},
```

```
        {0, 0, 0, 0}};
```

```
    int option_index = 0;
```

```
        c = getopt_long(argc, argv, "t:", long_options, &option_index);
```

```
        if (c == -1) break;
```

```
        switch (c)
```

```
        {
```

```
        case 't':    printf ("option -t /--testwith value `"%s"\n", optarg);
```

```
                    break;
```

```
        default:
```

```
                    abort ();
```

```
        }
```

```
    }
```

```
    return 0;
```

```
}
```

Техніка налагодження програм в ОС Linux за допомогою операторів друку **printf**, **snprintf**, **fprintf**

При розробці системних програм іноді виникають ситуації, в яких не доцільно користуватися програмами для налагодження типу **ddd**. Наприклад, це може бути у ситуації, коли виконання програми якимось чином залежить від плинності часу, і призупинення програми налагоджувачем призведе до спотворення результатів, неправильної подальшої поведінки програми, тощо. У таких ситуаціях має сенс використання функцій для форматowanego виводу інформації:

printf – форматований на екран;

snprintf – форматований у масив символів;

fprintf – форматований у файл.

Функція **printf** повинна бути вже відома студентам. Розглянемо дві інші функції.

int fprintf(FILE* stream, const char* template, ...);

Функція **fprintf** має із функцією **printf** однакову структуру рядку форматів **template** і відрізняється тільки тим, що виконує вивід до файлу **stream**.

int snprintf(char* s, size_t size, const char* template, ...);

Функція **snprintf** також має із функцією **printf** однакову структуру рядку форматів **template** і відрізняється тільки тим, що виконує вивід у рядок **s**. Крім того, ця функція перевіряє, щоб створений із **template** рядок не був більше розміру **size** (зазвичай **size** дорівнює розміру рядку **s**) і не викликає помилки сегментування пам'яті. Крім функції **snprintf** також існує її різновид – функція **sprintf** (у імені функції відсутня буква **n** після букви **s**), яка не виконує такої перевірки, і тому її використання не рекомендується.

«Найдешевшою» за часом виконання є функція **snprintf** тому, що вона виконує запис інформації у рядок, тобто у оперативну пам'ять. Техніка використання **snprintf** полягає у наступному:

1. Перед запуском основного алгоритму програми виділити пам'ять під вектор рядків фіксованої довжини;

2. Під час роботи алгоритму послідовно заносити у вектор повідомлення за допомогою функції **snprintf**;
3. Після закінчення роботи алгоритму виконати вивід інформації з цього вектора на екран або у файл за допомогою функцій **printf**, **fprintf**.

Створення найпростішої програми на мові Java в ОС Linux та Windows

Для створення найпростішої програми мовою Java необхідно створити файл з розширенням **.java** (наприклад, **Main.java**) та записати в нього наступні рядки:

```
class Main {  
    static public void main(String[] argv) {  
        System.out.println("Hello, world!");  
    }  
}
```

Для компіляції створеної програми необхідно виконати команду запуску Java-компілятора (**javac**):

```
$ javac Main.java
```

Після виконання вищенаведеної команди буде створений файл із ім'ям **main.class**. Цей файл є Java-програмою, що може виконуватись на віртуальній Java-машині. Для виконання цієї програми необхідно у директорії, де знаходиться файл-клас **main.class**, виконати наступну команду:

```
$ java Main
```

Цією командою ми вказуємо, що необхідно викликати метод **main** із класу **Main**. Метод **main** повинен бути оголошений із кваліфікаторами **public** та **static**, повинен не повертати жодного значення, і повинен приймати один параметр типу **String[]**.

Зверніть увагу, що дана програма не містить виразу **package**. Це означає, що даний клас буде знаходитися в “**default package**” – пакеті за замовчуванням. Це не є добре, і при виконанні лабораторної обов'язково необхідно розкласти всі класи по пакетах, як буде показано далі.

Створення бібліотеки (модуля) на мові Java

Типова бібліотека (модуль) на мові Java – це звичайний файл із класом, в якому немає методу **main**. Наприклад:

```
package pretty;

public class PrettyPrinter {
    public static void prettyPrint(String[] params) {

        if(params.length > 0) {
            System.out.println("Program argument list:");
        }

        for (int i = 0; i < params.length; i++) {
            System.out.println("Parameter #" + i + " = " + params[i]);
        }
    }
}
```

Початковий код мовою Java необхідно зберегти у файлі **PrettyPrinter.java**, а для компіляції необхідно виконати наступну команду:

```
$ javac -d . PrettyPrinter.java
```

В результаті буде створено директорію **pretty**, у якій буде збережено файл **PrettyPrinter.class**.

Тепер цю бібліотеку можна використовувати в інших Java-програмах, для чого необхідно імпортувати клас **PrettyPrinter** до цієї програми наступним чином:

```
import pretty.PrettyPrinter;
```

Система пакетів мови Java та компіляція багатофайлової програми

В Java є гнучка та потужна система пакетів, що дозволяє відмовитися від компіляції файлів окремо, і може автоматично шукати залежності.

Java трактує структуру директорій файлової системи із вхідними файлами **.java**, що містять програмний код мовою Java, як ієрархію пакетів. Тому надзвичайно важливо, щоб при написанні програми було коректно вказано директиву **package** — вона повинна відповідати тому місцю, де лежить файл відносно кореня проекту. Наприклад, для наступної ієрархії файлової системи:

```
.
├── pro
│   ├── lab2
│   │   ├── func
│   │   │   ├── Func.java
│   │   │   └── someExamplePackage
│   │   │       └── One.java
```

Файл **One.java** повинен містити:

```
package pro.lab2.someExamplePackage;
```

А файл **Func.java** повинен містити:

```
package pro.lab2.func;
```

При цьому для імпорту в **One.java** коду із **Func.java** необхідно в коді **One.java** додати:

```
import pro.lab2.func.Func;
```

Зверніть увагу, що останнім в імпорті записується *ім'я класу*, що імпортується.

При правильному вказанні пакетів та конструкцій імпорту, компіляція проводиться в одну команду, важливо запускати компіляцію із кореня проекту:

```
$ javac pro/lab2/someExamplePackage/One.java
```

Після цього, для запуску програми необхідно вказати із того ж кореня проекту наступну команду:

```
$ java pro.lab2.someExamplePackage.One
```

Зверніть увагу: при запуску програми вказується ім'я класу, як при імпорті. Тому ставиться крапка замість символу «/» (слеш), і немає розширення **.java** в кінці.

JAR архіви

Основним форматом для розповсюдження Java-програм є jar-архів (Java ARchive).

Для створення jar-архіву програми, яка складається із файлу головного класу **One.class** та допоміжної бібліотеки, що знаходиться у файлі **Func.class**, необхідно виконати наступну команду:

```
$ jar -cfm main.jar manifest pro/lab2/someExamplePackage/One.class  
pro/lab2/func/Func.class
```

Призначення використаних опцій:

- c – вказує на те, що створюється новий архів;
- f – вказує ім'я нового архіву;
- m – вказує ім'я файлу маніфесту, який буде додано до архіву.

Файл маніфесту використовується для оголошення головного класу програми і зазвичай складається із одного рядку:

```
Main-Class: pro.lab2.someExamplePackage.One
```

Цим вказується, що основним класом є клас **One**, у якому повинен бути оголошений статичний метод **main**. В кінці файлу маніфесту *обов'язково* повинен бути порожній рядок.

Зверніть увагу! До архіву пакуються згенеровані **.class** файли, а не вхідний код! Всі файли, що кладуться до архіву, повинні бути явно перераховані аргументами команди пакування **jar**.

Для запуску jar-архіву необхідно виконати наступну команду:

```
$ java -jar main.jar
```

Постановка завдання для програми мовою C

1. Написати програму розв'язання задачі пошуку (за варіантом) у двовимірному масиві (матриці) одним з алгоритмів методу лінійного пошуку.

Розміри матриці m та n взяти самостійно у межах від 7 до 10. Розмір матриці повинен задаватися аргументом запуску програми.

Програма **обов'язково** повинна бути написана і структурована таким чином:

- a) оголошення структур даних (**typedef**) повинно бути зроблено у окремому заголовочному файлі;
- b) повинно бути щонайменше три файли із вихідним кодом (не враховуючи необхідні заголовочні файли), що міститимуть реалізації функцій введення (випадкові значення, наперед сортовані значення, з клавіатури), обробки, та виведення на друк (**pretty_print**) елементів матриці;
- c) для виконання завдання обробки елементів матриці повинно бути написано дві різні функції:
 - 1) з додатковими операторами виведення налагоджувальної інформації на друк (**debug**-версія);
 - 2) з виконанням заданих дій без додаткового виведення налагоджувальної інформації (**release**-версія).

4. Вибір функції повинен робити користувач при запуску програми через аргумент запуску. Наприклад, опція **-d** вмикає **debug** режим.

5. Для компіляції написаної багатофайлової програми написати окремий **make**-файл, причому:

- a) при зміні одного із вихідних файлів повинен перекомпільовуватися лише цей файл (а також відбуватися дії, необхідні для генерації бінарного файлу);
- b) при видаленні бінарного файлу та незмінних вихідних файлах повинне відбуватися лише лінування (компоновка бінарного файлу з об'єктних);
- c) забезпечити окрему ціль для очистки згенерованих файлів;

6. Забезпечити можливість компіляції написаної багатофайлової програми двома способами:

- a) за допомогою однієї команди **gcc**;
- b) за допомогою **make**-файлу.

7. Виконати тестування та налагодження програми на комп'ютері. При тестуванні програми необхідно підбирати такі вхідні набори початкових значень матриці, щоб можна було легко відстежити коректність виконання пошуку і ця коректність була б протестована для всіх можливих випадків. З метою тестування дозволяється використовувати матриці меншого розміру.

Постановка завдання для програми мовою Java

1. Написати консольну програму розв'язання задачі пошуку (за варіантом) у двовимірному масиві (матриці) одним з алгоритмів методу лінійного пошуку.

Розміри матриці m та n взяти самостійно у межах від 7 до 10.

При написанні програми повинно бути реалізовано щонайменше три класи, один із яких буде відповідати за пошук елемента в матриці, другий відповідати за ввід-вивід матриці, а третій — за головний клас, що міститиме метод **main**.

Для компіляції та запуску написаної програми написати окремий **make**-файл, причому забезпечити окремі цілі для очистки згенерованих файлів, а також генерації **jar**-архіву.

Вміти компілювати написану програму двома способами:

- a) за допомогою однієї команди **javac**;
- b) за допомогою **make**-файлу.

6. Виконати тестування та налагодження програми на комп'ютері. При тестуванні програми необхідно підбирати такі вхідні набори початкових значень матриці, щоб можна було легко відстежити коректність виконання пошуку і ця коректність була б протестована для всіх можливих випадків. З метою тестування дозволяється використовувати матриці меншого розміру.

Зміст звіту

1. Загальна постановка задачі та завдання для конкретного варіанту.
2. Текст програми мовою C, вхідні дані.
3. Командні рядки для компілювання та запуску програми мовою C.
4. Make-файл для компілювання та запуску програми мовою C.
5. Текст програми мовою Java, вхідні дані.
6. Командні рядки для компілювання та запуску програми мовою Java.
7. Make-файл для компілювання та запуску програми мовою Java.
8. Тести для налагодження програми і результати, отримані для них на комп'ютері.

Вимоги до звіту

1. Код друкується моноширинним шрифтом.
 2. Код кожного файлу повинен бути відділений від інших суцільною лінією та мати підпис із назвою цього файлу. Приклад лінії:
-

Для її створення у більшості текстових редакторів достатньо на порожньому рядку набрати три чи п'ять дефісів: ----- та натиснути клавішу Enter.

3. Заборонено використовувати низькоякісні скріншоти завдання із методички. Для отримання якісного скріншоту достатньо відкрити файл **pdf** на весь екран і зробити скріншот побільше, потім стиснути і зберегти зображення у форматі **png**.

4. Нумерація сторінок є обов'язковою.

Контрольні питання

1. Призначення утиліти **make**.
2. Структура блока файлу **makefile**.
3. Послідовність обробки правил.
4. Фіктивні цілі.
5. Способи визначення необхідності повторного виконання правила.
6. Відповідність системи пакетів та їх імпорту в Java до файлової системи.

Варіанти індивідуальних завдань

Розміри матриці m і n взяти самостійно у межах від 7 до 10.

Варіант 1

Задано матрицю дійсних чисел $A[m][n]$. У кожному рядку матриці визначити присутність заданого дійсного числа X і його місцезнаходження (координати).

Варіант 2

Задано матрицю дійсних чисел $A[n][n]$. У головній діагоналі матриці знайти перший додатний і останній від'ємний елементи, а також поміняти їх місцями.

Варіант 3

Задано матрицю дійсних чисел $A[m][n]$. При обході матриці по рядках знайти в ній перший додатний елемент і його місцезнаходження (координати).

Варіант 4

Задано матрицю дійсних чисел $A[m][n]$. У кожному стовпчику матриці визначити присутність заданого дійсного числа X і його місцезнаходження (координати).

Варіант 5

Задано матрицю дійсних чисел $A[n][n]$. У побічній діагоналі матриці знайти перший мінімальний і останній максимальний елементи, а також поміняти їх місцями.

Варіант 6

Задано матрицю дійсних чисел $A[m][n]$. При обході матриці по стовпчиках знайти в ній перший мінімальний елемент і його місцезнаходження (координати).

Варіант 7

Задано матрицю дійсних чисел $A[m][n]$. У кожному рядку матриці знайти останній від'ємний елемент і його місцезнаходження (координати).

Варіант 8

Задано матрицю дійсних чисел $A[n][n]$. У головній діагоналі матриці знайти перший від'ємний і останній додатний елементи, а також поміняти їх місцями.

Варіант 9

Задано матрицю дійсних чисел $A[m][n]$. При обході матриці по рядках знайти в ній останній нульовий елемент і його місцезнаходження (координати).

Варіант 10

Задано матрицю дійсних чисел $A[m][n]$. У кожному стовпчику матриці знайти останній максимальний елемент і його місцезнаходження (координати).

Варіант 11

Задано матрицю дійсних чисел $A[n][n]$. У побічній діагоналі матриці знайти перший максимальний і останній мінімальний елементи, а також поміняти їх місцями.

Варіант 12

Задано матрицю дійсних чисел $A[m][n]$. При обході матриці по стовпчиках знайти в ній останній мінімальний елемент і його місцезнаходження (координати).

Варіант 13

Задано матрицю дійсних чисел $A[m][n]$. У кожному рядку матриці знайти перший максимальний елемент і його місцезнаходження (координати).

Варіант 14

Задано матрицю дійсних чисел $A[n][n]$. У головній діагоналі матриці знайти перший мінімальний і останній максимальний елементи, а також поміняти їх місцями.

Варіант 15

Задано матрицю дійсних чисел $A[m][n]$. При обході матриці по рядках визначити в ній присутність заданого дійсного числа X і його місцезнаходження (координати).

Варіант 16

Задано матрицю дійсних чисел $A[m][n]$. У кожному стовпчику матриці знайти перший від'ємний елемент і його місцезнаходження (координати).

Варіант 17

Задано матрицю дійсних чисел $A[n][n]$. У побічній діагоналі матриці знайти перший додатний і останній від'ємний елементи, а також поміняти їх місцями.

Варіант 18

Задано матрицю дійсних чисел $A[m][n]$. При обході матриці по стовпчиках визначити в ній присутність заданого дійсного числа X і його місцезнаходження (координати).

Варіант 19

Задано матрицю дійсних чисел $A[m][n]$. У кожному рядку матриці знайти перший нульовий елемент і його місцезнаходження (координати).

Варіант 20

Задано матрицю дійсних чисел $A[n][n]$. У головній діагоналі матриці знайти перший максимальний і останній мінімальний елементи, а також поміняти їх місцями.

Варіант 21

Задано матрицю дійсних чисел $A[m][n]$. При обході матриці по рядках знайти в ній останній максимальний елемент і його місцезнаходження (координати).

Варіант 22

Задано матрицю дійсних чисел $A[m][n]$. У кожному стовпчику матриці знайти останній додатний елемент і його місцезнаходження (координати).

Варіант 23

Задано матрицю дійсних чисел $A[n][n]$. У побічній діагоналі матриці знайти перший від'ємний і останній додатний елементи, а також поміняти їх місцями.

Варіант 24

Задано матрицю дійсних чисел $A[m][n]$. При обході матриці по стовпчиках знайти в ній перший від'ємний елемент і його місцезнаходження (координати).

Варіант 25

Задано матрицю дійсних чисел $A[m][n]$. У кожному рядку матриці знайти останній мінімальний елемент і його місцезнаходження (координати).

Варіант 26

Задано матрицю дійсних чисел $A[m][n]$. При обході матриці по стовпчиках знайти в ній останній додатний елемент і його місцезнаходження (координати).

Варіант 27

Задано матрицю дійсних чисел $A[m][n]$. При обході матриці по рядках знайти в ній перший максимальний елемент і його місцезнаходження (координати).

Варіант 28

Задано матрицю дійсних чисел $A[m][n]$. У кожному стовпчику матриці знайти перший мінімальний елемент і його місцезнаходження (координати).

Варіант 29

Задано матрицю дійсних чисел $A[n][n]$. У побічній діагоналі матриці визначити присутність заданого дійсного числа X і його місцезнаходження (координати).

Варіант 30

Задано матрицю дійсних чисел $A[m][n]$. При обході матриці по рядках знайти в ній останній мінімальний елемент і його місцезнаходження (координати).

ЛАБОРАТОРНА РОБОТА №2. СТВОРЕННЯ ПАРАЛЕЛЬНИХ ПОТОКІВ ОПЕРАЦІЙНОЇ СИСТЕМИ LINUX ТА НАЙПРОСТІША СИНХРОНІЗАЦІЯ

Постановка задачі

1. Опрацювати всі надані лектором приклади коду паралельних потоків по темі «Засоби взаємодії паралельних потоків операційної системи Linux», що знаходяться в директорії **01_Creation**, тобто:

- вміти запускати всі ці приклади і отримувати результати на захисті лабораторної роботи;
- знати які структури даних та функції взаємодії паралельних потоків описані в коді кожного прикладу та як вони працюють, а також вміти це пояснити на захисті лабораторної роботи;
- бути готовими до виконання модифікацій будь-яких з цих прикладів на захисті лабораторної роботи.

2. Написати програму, яка виводить на друк таблицю значень трьох функцій при паралельній реалізації обчислення значень кожної функції за допомогою трьох паралельних потоків згідно наведених нижче вимог.

3. Кожна із трьох заданих функцій $f_1(x)$, $f_2(x)$, $f_3(x)$ обчислює свої значення при зміні значень аргументу x_i ($i=0, 1, \dots, n$; $n>0$) в межах діапазону, заданого дійсними числами a та b ($b>a$); тобто значення x_i обчислюються за формулою $x_i=a+ih$, де $h=(b-a)/n$.

4. Отримані дійсні значення x_i та $f_1(x_i)$, $f_2(x_i)$, $f_3(x_i)$ вивести на екран у вигляді таблиці заданої форми (табл. 1).

5. Номери функцій $f_1(x)$, $f_2(x)$, $f_3(x)$, а також значення a , b , n визначаються за варіантом завдання (табл. 2).

6. Виконати налагодження написаної програми.

Вимоги до програми

1. Програма повинна правильно розв'язувати поставлену задачу при вхідних даних a , b , n ($a < b$, $n \leq 10$).

2. В програмі не дозволяється використовувати масиви ані для збереження обчислених значень функцій, ані для будь-яких інших цілей.

3. В заголовку надрукованої таблиці мають вказуватися назви функції відповідно до варіанта, наприклад, $SIN(x)$, $ABS(x+7)*5$, а не $f_1(x)$, $f_2(x)$, $f_3(x)$.

4. Обчислення значень математичних функцій $f_1(x)$, $f_2(x)$, $f_3(x)$ мусить відбуватися з врахуванням області допустимих значень **в рамках трьох паралельних потоків**.

5. Створення та запуск усіх трьох потоків, що обчислюють значення математичних функцій $f_1(x)$, $f_2(x)$, $f_3(x)$, повинні бути виконані у головній програмі (головному, четвертому, потоці). **При створенні потоків їм повинні бути передані значення a , b , n через аргумент потоку**.

6. Виведення результуючої таблиці заданої форми (табл. 1) на екран повинно бути виконано у головному (четвертому) потоці багатопоточної програми.

7. Алгоритм кожного з трьох паралельних потоків, що обчислюють значення математичних функцій $f_1(x)$, $f_2(x)$, $f_3(x)$, повинен бути реалізований у вигляді циклу, що обчислює задану кількість значень функції згідно заданих значень a , b , n .

7. Передавання значень функцій $f_1(x)$, $f_2(x)$, $f_3(x)$, що обчислюються у трьох потоках, до головного потоку виконувати після отримання кожного нового значення функцій через глобальні змінні.

8. Для синхронізації обчислення значень функцій $f_1(x)$, $f_2(x)$, $f_3(x)$ у трьох потоках та виведення на екран нового рядка таблиці після отримання нових значень функцій у головному потоці дозволяється використовувати тільки конструкцію бар'єра (*`pthread_barrier_t`*) та затримки (функція *`usleep()`*).

Зміст звіту

1. Постановка задачі, вимоги до програми та конкретний варіант завдання.
2. Текст програми.
3. Тести для налагодження і результати налагодження, отримані на комп'ютері.
4. Побудована таблиця значень функцій $f_1(x)$, $f_2(x)$, $f_3(x)$.

Таблиця 1

Функції однієї змінної			
Аргумент x	Назва функції $f_1(x)$	Назва функції $f_2(x)$	Назва функції $f_3(x)$
x_0	$f_1(x_0)$	$f_2(x_0)$	$f_3(x_0)$
x_1	$f_1(x_1)$	$f_2(x_1)$	$f_3(x_1)$
x_2	$f_1(x_2)$	$f_2(x_2)$	$f_3(x_2)$
...
x_n	$f_1(x_n)$	$f_2(x_n)$	$f_3(x_n)$

Склав: Прізвище, ініціали, група

Варіанти завдань

У кожному варіанті таблиці 2 вказані значення a , b , n , а також номери функцій $f_1(x)$, $f_2(x)$, $f_3(x)$, які потрібно взяти з таблиці 3.

Таблиця 2

№ п/п	Номери функцій $f_1(x), f_2(x),$ $f_3(x)$	a	b	n	№ п/п	Номери функцій $f_1(x), f_2(x),$ $f_3(x)$	a	b	n
1	1, 5, 9	0	2π	8	16	1, 7, 13	0	2π	8
2	2, 6, 10	$-\pi$	π	10	17	2, 8, 14	$-\pi$	π	10
3	3, 7, 11	-2π	0	12	18	3, 9, 13	-2π	0	12
4	4, 8, 13	0	π	8	19	4, 5, 11	0	π	8
5	5, 9, 14	$-\pi/2$	$\pi/2$	10	20	7, 10, 11	$-\pi/2$	$\pi/2$	10
6	1, 9, 12	0	2π	12	21	4, 9, 11	0	2π	12
7	2, 5, 11	$-\pi$	π	8	22	3, 8, 11	$-\pi$	π	8
8	3, 6, 12	-2π	0	10	23	8, 10, 14	-2π	0	10
9	4, 7, 14	0	π	12	24	4, 10, 12	0	π	12
10	5, 10, 13	$-\pi/2$	$\pi/2$	8	25	8, 9, 12	$-\pi/2$	$\pi/2$	8
11	1, 8, 12	0	2π	10	26	1, 6, 11	0	2π	10
12	2, 9, 13	$-\pi$	π	12	27	7, 9, 13	$-\pi$	π	12
13	3, 5, 14	-2π	0	8	28	2, 7, 12	-2π	0	8
14	4, 6, 9	0	π	10	29	7, 10, 14	0	π	10
15	7, 9, 12	$-\pi/2$	$\pi/2$	12	30	3, 10, 11	$-\pi/2$	$\pi/2$	12

Таблиця 3

Номер функції	Функція	Номер функції	Функція
1.	$\sin^2(x) + \cos(x)$	8.	$\cos^2(x) * (1 - \sin(x))$
2.	$\cos^2(x) + \sin(x)$	9.	$\sin(x) - \cos^2(x)$
3.	$\sin^2(x) * \cos(x)$	10.	$\cos(x) - \sin^2(x)$
4.	$\cos^2(x) * \sin(x)$	11.	$\sin(x) * (1 + \cos^2(x))$
5.	$\sin^2(x) * (1 + \cos(x))$	12.	$\cos(x) * (1 + \sin^2(x))$
6.	$\cos^2(x) * (1 + \sin(x))$	13.	$\sin(x) * (1 - \cos^2(x))$
7.	$\sin^2(x) * (1 - \cos(x))$	14.	$\cos(x) * (1 - \sin^2(x))$

Контрольні питання

1. Стани паралельних процесів та потоків у операційній системі Linux.
2. Для чого використовується тип **pthread_t**?
3. Які є обмеження на заголовок потокової функції?
4. Функція **pthread_create()** та її параметри.
5. Напишіть команду компіляції файлу з багатопотоковою програмою.
6. Як виконати чекання на завершення певного потоку?
7. Як передати декілька значень через аргумент потоку?
8. Як повернути результат роботи певного потоку у інший потік через значення, що повертається потоковою функцією?
9. Як працює засіб синхронізації «бар'єр»?
10. Як оголосити, проініціалізувати та використовувати бар'єр?

ЛАБОРАТОРНА РОБОТА №3. ЗАСОБИ ВЗАЄМОДІЇ ПАРАЛЕЛЬНИХ ПОТОКІВ ОПЕРАЦІЙНОЇ СИСТЕМИ LINUX

Постановка задачі

1. Опрацювати всі надані лектором приклади коду паралельних потоків по темі «Засоби взаємодії паралельних потоків операційної системи Linux», що знаходяться в директоріях **02_Threads_ID** та **03_Programs_to_Schemas_from_Conspect**, тобто:
 - вміти запускати всі ці приклади і отримувати результати на захисті лабораторної роботи;
 - знати які структури даних та конструкції взаємодії паралельних потоків описані в коді кожного прикладу та як вони працюють, а також вміти це пояснити на захисті лабораторної роботи;
 - розібратися з теоретичними ситуаціями, які відображують дані приклади, а також вміти їх розказати та пояснити на захисті лабораторної роботи;
2. Бути готовими до виконання модифікацій будь-яких з цих прикладів на захисті лабораторної роботи.

Контрольні питання

1. Напишіть команду компіляції файлу з багатопотоковою програмою та запустіть програму на виконання.
2. Що таке ідентифікатор процесу і як його можна отримати у програмі?
3. Що таке ідентифікатор потоку і як його можна отримати у програмі?
4. Дві головні проблеми при рішенні підзадачі синхронізації.
5. Завдання взаємного виключення: два підходи до його рішення.
6. В чому полягає підхід рішення завдання взаємного виключення, що ґрунтується на контролі процесів/потоків?
7. В чому полягає підхід рішення завдання взаємного виключення, що ґрунтується на контролі спільного ресурсу?
8. Завдання (проблема) несинхронної роботи без обміну інформації процесів/потоків.
9. Що таке атомарні структури даних і атомарні операції та які проблеми вони вирішують?

10. Поняття семафора, види семафорів та алгоритми виконання операцій над семафорами.
11. Застосування двійкового семафору для рішення завдання взаємного виключення: схема, пояснення, описи даних та функції роботи із семафорами бібліотеки **<semaphore.h>**.
12. Застосування двійкового семафору для рішення завдання простої (неповної) синхронізації: схема, пояснення та функції роботи із семафорами бібліотеки **<semaphore.h>**.
13. Застосування двійкових семафорів для рішення завдання повної синхронізації: схема, пояснення та функції роботи із семафорами бібліотеки **<semaphore.h>**.
14. Поняття м'ютекса та алгоритми виконання операцій над м'ютексом.
15. Застосування м'ютекса для рішення завдання взаємного виключення: схема, пояснення, описи даних та функції роботи із м'ютексами бібліотеки **<pthread.h>**.
16. Поняття сигнальних (умовних) змінних, їх види та алгоритми виконання операцій над сигнальними (умовними) змінними.
17. Застосування умовних змінних бібліотеки **<pthread.h>** для рішення завдання повної синхронізації: схема, пояснення та функції роботи із семафорами бібліотеки **<pthread.h>**. Як правильно використовувати умовні змінні бібліотеки **<pthread.h>** і в чому полягає можлива некоректність їх використання?
18. Поняття бар'єра та особливості використання бар'єрів бібліотеки **<pthread.h>**.
19. Застосування бар'єра для рішення завдання повної синхронізації: схема, пояснення та функції роботи із бар'єрами бібліотеки **<pthread.h>**.

ЛАБОРАТОРНА РОБОТА №4. КОМПЛЕКСНЕ ВИКОРИСТАННЯ ЗАСОБІВ ВЗАЄМОДІЇ ПАРАЛЕЛЬНИХ ПОТОКІВ ОПЕРАЦІЙНОЇ СИСТЕМИ LINUX

Постановка завдання та вимоги до виконання програми

1. Опрацювати всі надані лектором приклади коду паралельних потоків по темі «Засоби взаємодії паралельних потоків операційної системи Linux», що знаходяться в директоріях **04_Common_Resource** та **05_Atomic_Operations**, тобто:
 - вміти запускати всі ці приклади і отримувати результати на захисті лабораторної роботи;
 - знати які структури даних та конструкції взаємодії паралельних потоків описані в коді кожного прикладу та як вони працюють, а також вміти це пояснити на захисті лабораторної роботи;
 - розібратися з теоретичними ситуаціями, які відображують дані приклади, а також вміти їх розказати та пояснити на захисті лабораторної роботи;
 - бути готовими до виконання модифікацій будь-яких з цих прикладів на захисті лабораторної роботи.
2. Написати програму, яка реалізує роботу паралельних потоків згідно заданої за варіантом схеми. Особливості реалізації синхронізації паралельних потоків та взаємного виключення потоків при доступі до спільних ресурсів задані за варіантами у таблицях 1 та 2.
3. При написанні програми виконати повне трасування роботи програми за допомогою операторів друку, тобто розставити в програмі оператори друку таким чином, щоб можна було прослідкувати всі варіанти виконання паралельних потоків і впевнитись у коректності роботи програми. Протокол трасування рекомендується записувати у файл (log-файл).
4. Запуск усіх потоків повинен бути виконаний у головній програмі.
5. Кожен потік повинен бути організованим у вигляді нескінченного циклу.
6. Всі дії задані за варіантами, що вказані у таблиці, повинні бути виконані всередині цього нескінченного циклу.
7. Взаємне розташування операторів синхронізації та доступу до спільного ресурсу, якщо вони знаходяться у одному потоці, є довільним.

8. Оскільки синхронізація за допомогою семафорів SCR21, SCR22 згідно завдання розташована всередині нескінченних циклів, то відразу після виконання синхронізації ці семафори повинні бути знову встановлені у початковий закритий стан.
9. Закінчення програми можна виконати двома способами:
 - примусовим перериванням за допомогою натиснення комбінації клавіш Ctrl+C;
 - оператором break при виконанні умови, яка стає істинною, коли буфер спільного ресурсу повністю заповнюється і повністю звільняється мінімум по два рази.
10. Якщо при реалізації паралельних потоків була використана функція usleep(), то передбачити режим запуску програми з «відключеними» функціями usleep().
11. Виконати налагодження написаної програми.

Зміст звіту

1. Загальна постановка завдання.
2. Завдання конкретного варіанту.
3. Схема паралельних потоків свого варіанту.
4. Текст програми.
5. Декілька протоколів роботи програми, які демонструють різні випадки роботи паралельних потоків.

Контрольні питання

1. Вміти відповідати на будь-які питання стосовно задачі Producer-Consumer (Постачальник-Споживач) та всіх структур даних, що використовуються в програмі (черга, стек, циклічний буфер тощо).
2. Знати принципи організації та засоби взаємодії паралельних потоків із загальнотеоретичної точки зору згідно теоретичній частині лекційного матеріалу, а також реалізацію цих засобів у операційній системі Linux щонайменше на прикладах, наданих до лекційного матеріалу.

Пояснення до таблиці з варіантами завдань

1. Потоків P1–P6 повинні бути організовані у вигляді нескінченних циклів, в тілі яких повинні бути реалізовані всі дії, які задані за варіантами завдань.
2. **CR1** – перший спільний ресурс (common resource) у вигляді буфера для обміну даними між потоками-постачальниками і потоками-споживачами P1–P6. Спосіб реалізації буфера та спосіб взаємного виключення потоків P1–P6 при доступі до CR1 визначається у таблиці 1 за варіантами.
Вид доступу до спільного ресурсу CR1 (запис чи читання) визначається типом кожного з потоків P1–P6 (постачальник чи споживач) і показаний на схемі за варіантом стрілочками.
Спосіб реалізації як семафора SCR1, так і м'ютекса MCR1 (блокуючий чи неблокуючий) вказані у таблиці завдань за варіантами.
3. **CR2** – другий спільний ресурс (common resource) у вигляді атомарних змінних.

Умовні терміни:

«Використання» – взяття значень атомарних змінних без їх зміни (наприклад, `__sync_fetch_and_add` з нулем) і використання цих значень для локальних обчислень;

«Модифікація» – зміна значень атомарних змінних без використання отриманих значень для локальних обчислень;

«Використання та модифікація» – зміна значень атомарних змінних і використання повернутих функціями значень для локальних обчислень.

4. У колонках «Засоби синхронізації паралельних потоків» таблиці варіантів завдань вказане завдання з синхронізації потоків. Для синхронізації потоків, в залежності від задачі синхронізації, яка вирішується (повна чи неповна) за варіантом, повинні бути використані:
 - для задачі повної синхронізації: або два двійкових семафори (SCR21+SCR22), або дві сигнальні (умовні) змінні (Sig21+Sig22), або бар'єр (BCR2);
 - для задачі неповної синхронізації: або один двійковий семафор SCR1, або одна сигнальна (умовна) змінна Sig21.Крім того, для кожного із семафорів SCR21 та SCR22 у таблиці вказаний спосіб його реалізації (блокуючий чи неблокуючий), а для сигнальних (умовних) змінних – вид сигналу (багатозначний чи одиничний).

5. ДСД – Динамічна Структура Даних.

Варіанти завдань

Таблиця 1

№ варіанту	№ схеми	Спільний ресурс 1 CR1 (буфер обміну даними)		Спільний ресурс 2 CR2	Засоби синхронізації паралельних потоків				
		Структура даних, що використовується у якості спільного ресурсу 1 (CR1)	Засоби взаємного виключення при доступі до спільного ресурсу 1 SCR1 + MCR1 або Sig1+Sig2+MCR1		Семафори та бар'єр для повної або неповної синхронізації потоків			Сигнальні (умовні) змінні синхронізації потоків	
					Атомарні дані (взяти по 2 змінних кожного з типів: <i>int, unsigned, long, long unsigned</i>) та операції (табл. 2)	Вид двійкового семафору SCR21	Вид двійкового семафору SCR22	Бар'єр BCR2	Вид сигналу сигнальної (умовної) змінної Sig21
1	5	Черга (ДСД)	Дві сигнальні (умовні) змінні Sig1 та Sig2 і блокуючий м'ютекс MCR1	1, 7, 3, 4, 11, 12, 13, 14	Блокуючий	—	—	Багатозначний	Одиничний
2	6	Стек (ДСД)	Блокуючий багатозначний семафор SCR1 та неблокуючий м'ютекс MCR1	1, 8, 3, 5, 10, 12, 13, 14	Неблокуючий	—	—	Одиничний	Одиничний
3	7	Цикл.буфер (ДСД)	Дві сигнальні (умовні) змінні Sig1 та Sig2 і неблокуючий м'ютекс MCR1	2, 7, 3, 6, 10, 11, 13, 14	Блокуючий	—	—	Одиничний	Одиничний
4	1	Стек (Вектор)	Дві сигнальні (умовні) змінні Sig1 та Sig2 і блокуючий м'ютекс MCR1	2, 8, 4, 5, 9, 12, 13, 14	Блокуючий	Блокуючий	—	Багатозначний	—
5	2	Цикл.буфер (Вектор)	Блокуючий багатозначний семафор SCR1 та блокуючий м'ютекс MCR1	1, 7, 4, 6, 9, 11, 13, 14	—	—	BCR2	Одиничний (P3) Одиничний (P6)	—
6	3	Черга (ДСД)	Дві сигнальні (умовні) змінні Sig1 та Sig2 і неблокуючий м'ютекс MCR1	2, 8, 5, 6, 9, 10, 13, 14	Блокуючий	Неблокуючий	—	Багатозначний	—
7	4	Стек (ДСД)	Блокуючий багатозначний семафор SCR1 та неблокуючий м'ютекс MCR1	1, 8, 3, 4, 11, 12, 13, 14	—	—	BCR2	Багатозначний	Багатозначний

8	8	Цикл.буфер (ДСД)	Неблокуючий багатозначний семафор SCR1 та блокуючий м'ютекс MCR1	2, 7, 3, 5, 10, 12, 13, 14	Неблокуючий	—	—	Одиничний	Багатозначний
9	9	Стек (Вектор)	Дві сигнальні (умовні) змінні Sig1 та Sig2 і неблокуючий м'ютекс MCR1	2, 8, 3, 6, 10, 11, 13, 14	Блокуючий	—	—	Багатозначний	Одиничний
10	10	Цикл.буфер (Вектор)	Блокуючий багатозначний семафор SCR1 та блокуючий м'ютекс MCR1	1, 7, 4, 5, 9, 12, 13, 14	Неблокуючий	—	—	Багатозначний	Багатозначний
11	1	Черга (ДСД)	Неблокуючий багатозначний семафор SCR1 та блокуючий м'ютекс MCR1	1, 8, 4, 6, 9, 11, 13, 14	—	—	VCR2	Багатозначний	—
12	2	Стек (ДСД)	Дві сигнальні (умовні) змінні Sig1 та Sig2 і блокуючий м'ютекс MCR1	1, 7, 5, 6, 9, 10, 13, 14	Неблокуючий	Блокуючий	—	Одиничний (P3) Багатозначний (P6)	—
13	3	Цикл.буфер (ДСД)	Неблокуючий багатозначний семафор SCR1 та неблокуючий м'ютекс MCR1	2, 7, 3, 4, 11, 12, 13, 14	—	—	VCR2	Багатозначний	—
14	4	Стек (Вектор)	Дві сигнальні (умовні) змінні Sig1 та Sig2 і неблокуючий м'ютекс MCR1	2, 8, 3, 5, 10, 12, 13, 14	Неблокуючий	Неблокуючий	—	Багатозначний	Одиничний
15	5	Цикл.буфер (Вектор)	Блокуючий багатозначний семафор SCR1 та блокуючий м'ютекс MCR1	1, 7, 3, 6, 10, 11, 13, 14	Неблокуючий	—	—	Одиничний	Одиничний
16	6	Черга (ДСД)	Дві сигнальні (умовні) змінні Sig1 та Sig2 і блокуючий м'ютекс MCR1	1, 8, 4, 5, 9, 12, 13, 14	Блокуючий	—	—	Багатозначний	Одиничний
17	7	Стек (ДСД)	Неблокуючий багатозначний семафор SCR1 та неблокуючий м'ютекс MCR1	2, 7, 4, 6, 9, 11, 13, 14	Неблокуючий	—	—	Одиничний	Одиничний
18	1	Цикл.буфер (ДСД)	Дві сигнальні (умовні) змінні Sig1 та Sig2 і неблокуючий м'ютекс MCR1	1, 8, 5, 6, 9, 10, 13, 14	Неблокуючий	Неблокуючий	—	Одиничний	—
19	2	Стек (Вектор)	Блокуючий багатозначний семафор SCR1 та неблокуючий м'ютекс MCR1	2, 8, 3, 4, 11, 12, 13, 14	—	—	VCR2	Одиничний (P3) Багатозначний (P6)	—

20	3	Цикл.буфер (Вектор)	Дві сигнальні (умовні) змінні Sig1 та Sig2 і блокуючий м'ютекс MCR1	1, 7, 3, 5, 10, 12, 13, 14	Неблокуючий	Блокуючий	—	Одиничний	—
21	4	Черга (ДСД)	Неблокуючий багатозначний семафор SCR1 та блокуючий м'ютекс MCR1	1, 8, 3, 6, 10, 11, 13, 14	—	—	VCR2	Одиничний	Багатозначний
22	8	Стек (ДСД)	Дві сигнальні (умовні) змінні Sig1 та Sig2 і блокуючий м'ютекс MCR1	2, 7, 4, 5, 9, 12, 13, 14	Блокуючий	—	—	Одиничний	Одиничний
23	9	Цикл.буфер (ДСД)	Блокуючий багатозначний семафор SCR1 та блокуючий м'ютекс MCR1	2, 8, 4, 6, 9, 11, 13, 14	Неблокуючий	—	—	Багатозначний	Одиничний
24	10	Стек (Вектор)	Дві сигнальні (умовні) змінні Sig1 та Sig2 і неблокуючий м'ютекс MCR1	2, 7, 5, 6, 9, 10, 13, 14	Блокуючий	—	—	Багатозначний	Одиничний
25	1	Цикл.буфер (Вектор)	Неблокуючий багатозначний семафор SCR1 та неблокуючий м'ютекс MCR1	1, 7, 3, 4, 9, 10, 13, 14	—	—	VCR2	Одиничний	—
26	2	Черга (ДСД)	Дві сигнальні (умовні) змінні Sig1 та Sig2 і неблокуючий м'ютекс MCR1	1, 8, 3, 5, 9, 11, 13, 14	Неблокуючий	Блокуючий	—	Одиничний (P3) Одиничний (P6)	—
27	3	Стек (ДСД)	Блокуючий багатозначний семафор SCR1 та блокуючий м'ютекс MCR1	2, 7, 3, 6, 9, 12, 13, 14	—	—	VCR2	Багатозначний	—
28	4	Цикл.буфер (ДСД)	Дві сигнальні (умовні) змінні Sig1 та Sig2 і блокуючий м'ютекс MCR1	2, 8, 4, 5, 10, 11, 13, 14	Блокуючий	Блокуючий	—	Багатозначний	Багатозначний
29	5	Стек (Вектор)	Дві сигнальні (умовні) змінні Sig1 та Sig2 і неблокуючий м'ютекс MCR1	1, 8, 4, 6, 10, 12, 13, 14	Неблокуючий	—	—	Багатозначний	Одиничний
30	6	Цикл.буфер (Вектор)	Неблокуючий багатозначний семафор SCR1 та блокуючий м'ютекс MCR1	2, 7, 5, 6, 11, 12, 13, 14	Блокуючий	—	—	Одиничний	Одиничний
31	7	Черга (ДСД)	Дві сигнальні (умовні) змінні Sig1 та Sig2 і блокуючий м'ютекс MCR1	1, 7, 5, 6, 11, 12, 13, 14	Неблокуючий	—	—	Одиничний	Одиничний

32	8	Стек (Вектор)	Блокуючий багатозначний семафор SCR1 та неблокуючий м'ютекс MCR1	2, 8, 3, 4, 9, 10, 13, 14	Блокуючий	—	—	Одиничний	Одиничний
33	9	Цикл.буфер (Вектор)	Дві сигнальні (умовні) змінні Sig1 та Sig2 і блокуючий м'ютекс MCR1	1, 8, 3, 6, 9, 12, 13, 14	Неблокуючий	—	—	Одиничний	Одиничний
34	10	Стек (ДСД)	Неблокуючий багатозначний семафор SCR1 та неблокуючий м'ютекс MCR1	2, 7, 4, 5, 10, 11, 13, 14	Блокуючий	—	—	Одиничний	Багатозначний

Таблиця 2 містить перелік вбудованих функцій, які реалізують атомарні операції.

Таблиця 2

Номер атомарної операції (функції)	Ідентифікатор вбудованої функції, що відповідає атомарній операції
1.	<i>type __atomic_add_fetch (type *ptr, type val, int memorder)</i>
2.	<i>type __atomic_sub_fetch (type *ptr, type val, int memorder)</i>
3.	<i>type __atomic_and_fetch (type *ptr, type val, int memorder)</i>
4.	<i>type __atomic_xor_fetch (type *ptr, type val, int memorder)</i>
5.	<i>type __atomic_or_fetch (type *ptr, type val, int memorder)</i>
6.	<i>type __atomic_nand_fetch (type *ptr, type val, int memorder)</i>
7.	<i>type __atomic_fetch_add (type *ptr, type val, int memorder)</i>
8.	<i>type __atomic_fetch_sub (type *ptr, type val, int memorder)</i>
9.	<i>type __atomic_fetch_and (type *ptr, type val, int memorder)</i>
10.	<i>type __atomic_fetch_xor (type *ptr, type val, int memorder)</i>
11.	<i>type __atomic_fetch_or (type *ptr, type val, int memorder)</i>
12.	<i>type __atomic_fetch_nand (type *ptr, type val, int memorder)</i>
13.	<i>bool __atomic_compare_exchange_n (type *ptr, type *expected, type desired, bool weak, int success_memorder, int failure_memorder)</i>
14.	<i>void __atomic_exchange (type *ptr, type *val, type *ret, int memorder)</i>

Схеми паралельних потоків за варіантами

Зауваження. На схемах використання двох подвійних ліній (верхньої та нижньої) означає, що дії між цими лініями повторюються циклічно. Тобто подвійні лінії використовуються аналогічно їх використанню в діаграмах дій.

СХЕМА 1

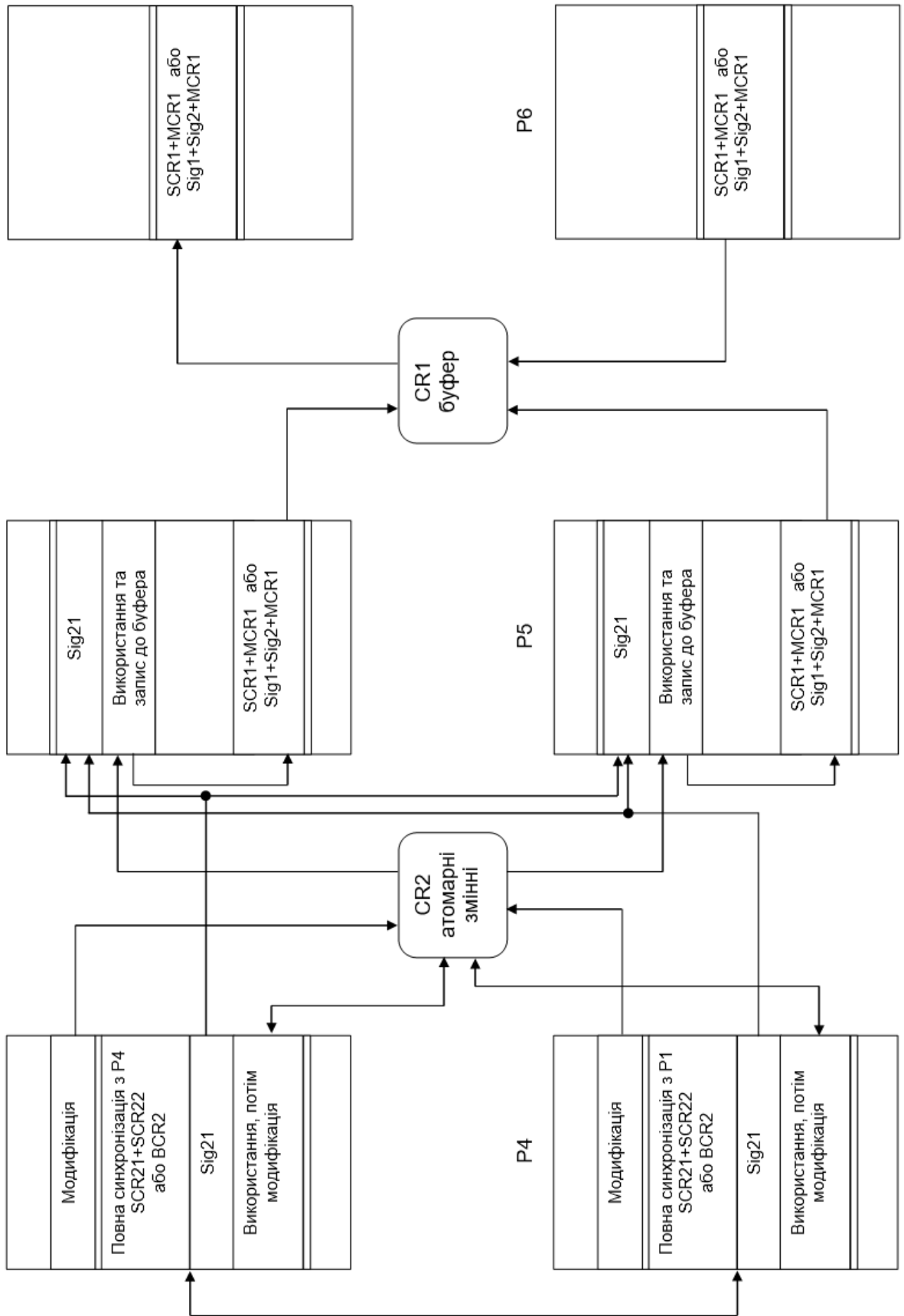


СХЕМА 2

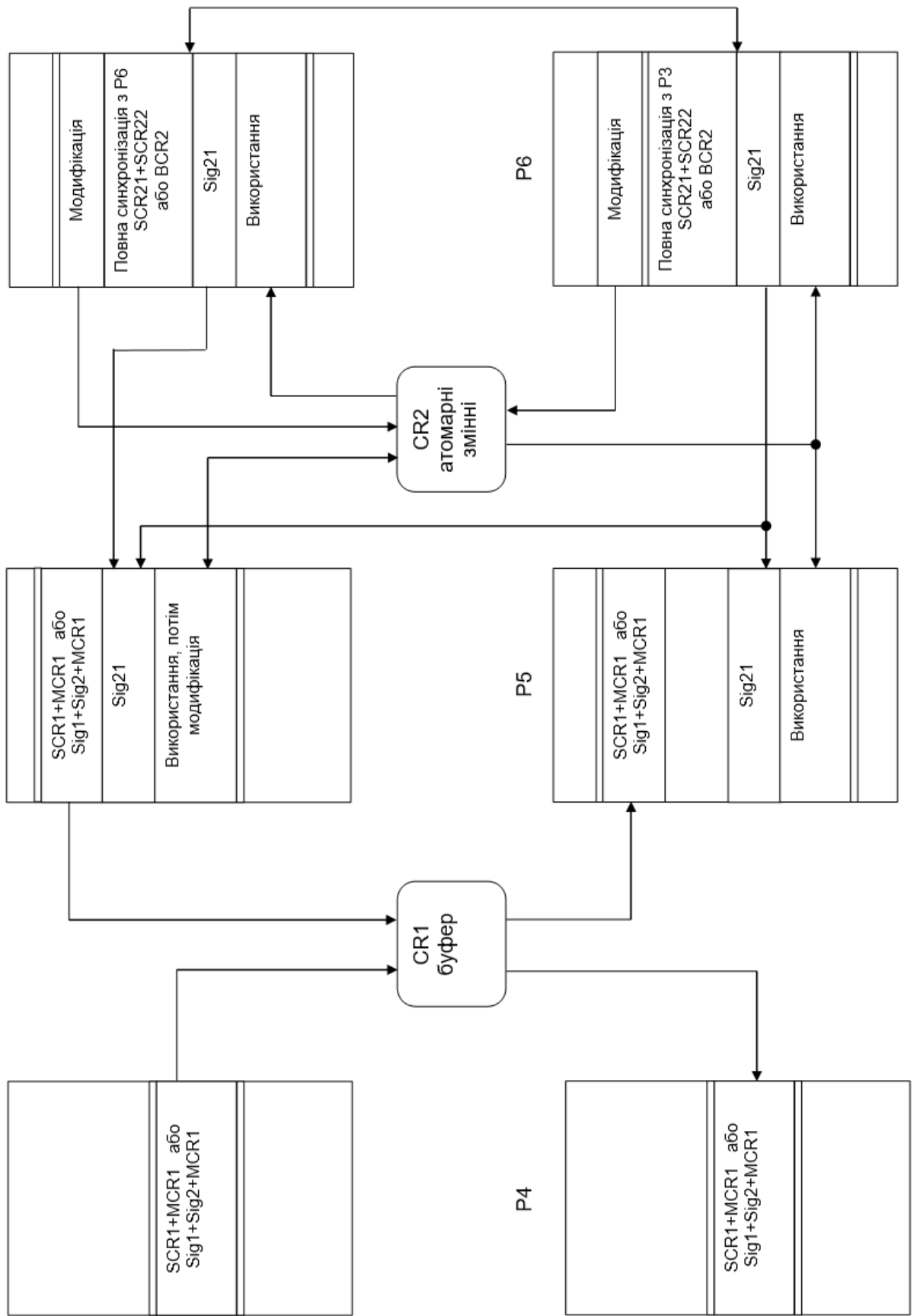


СХЕМА 3

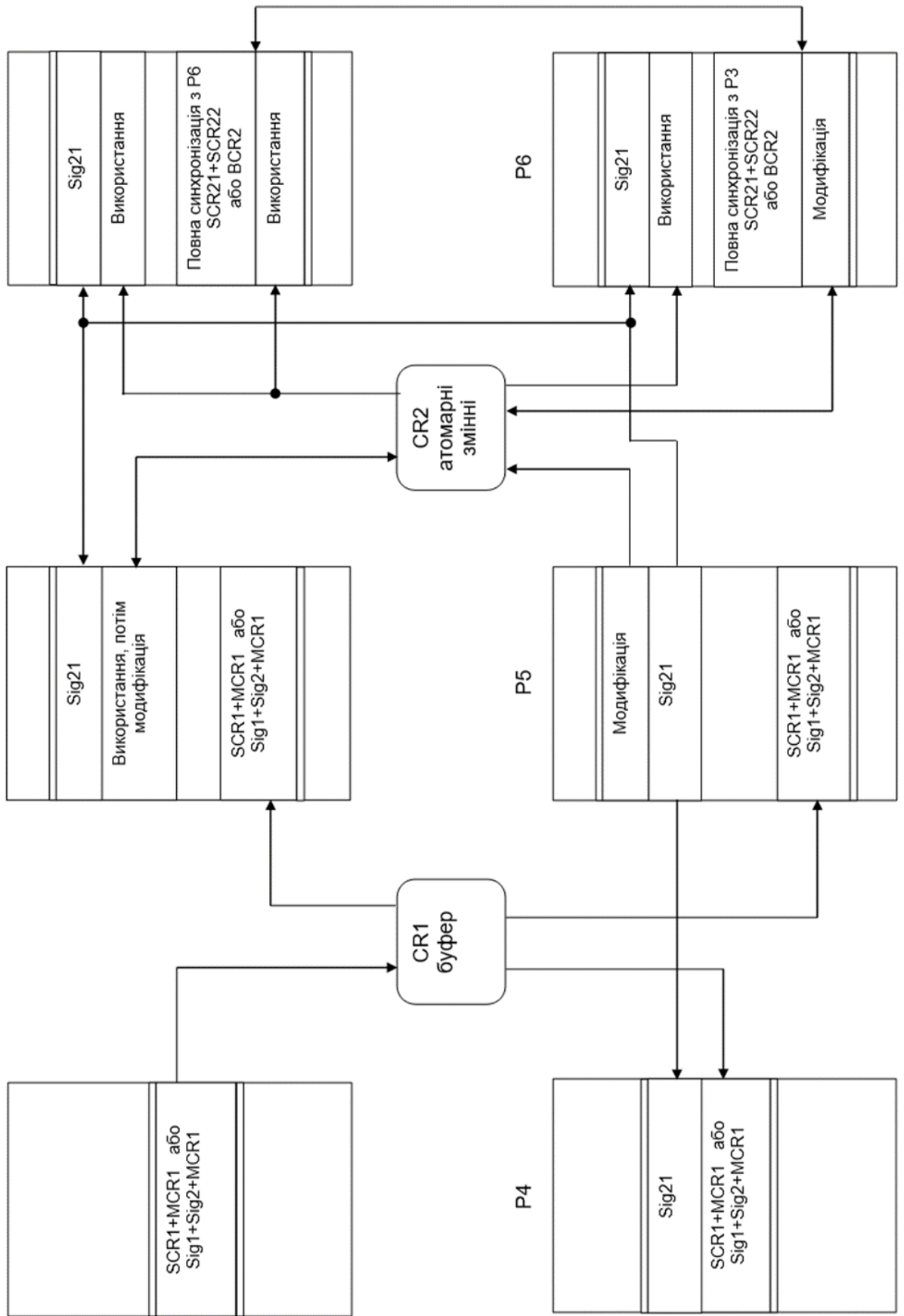


СХЕМА 4

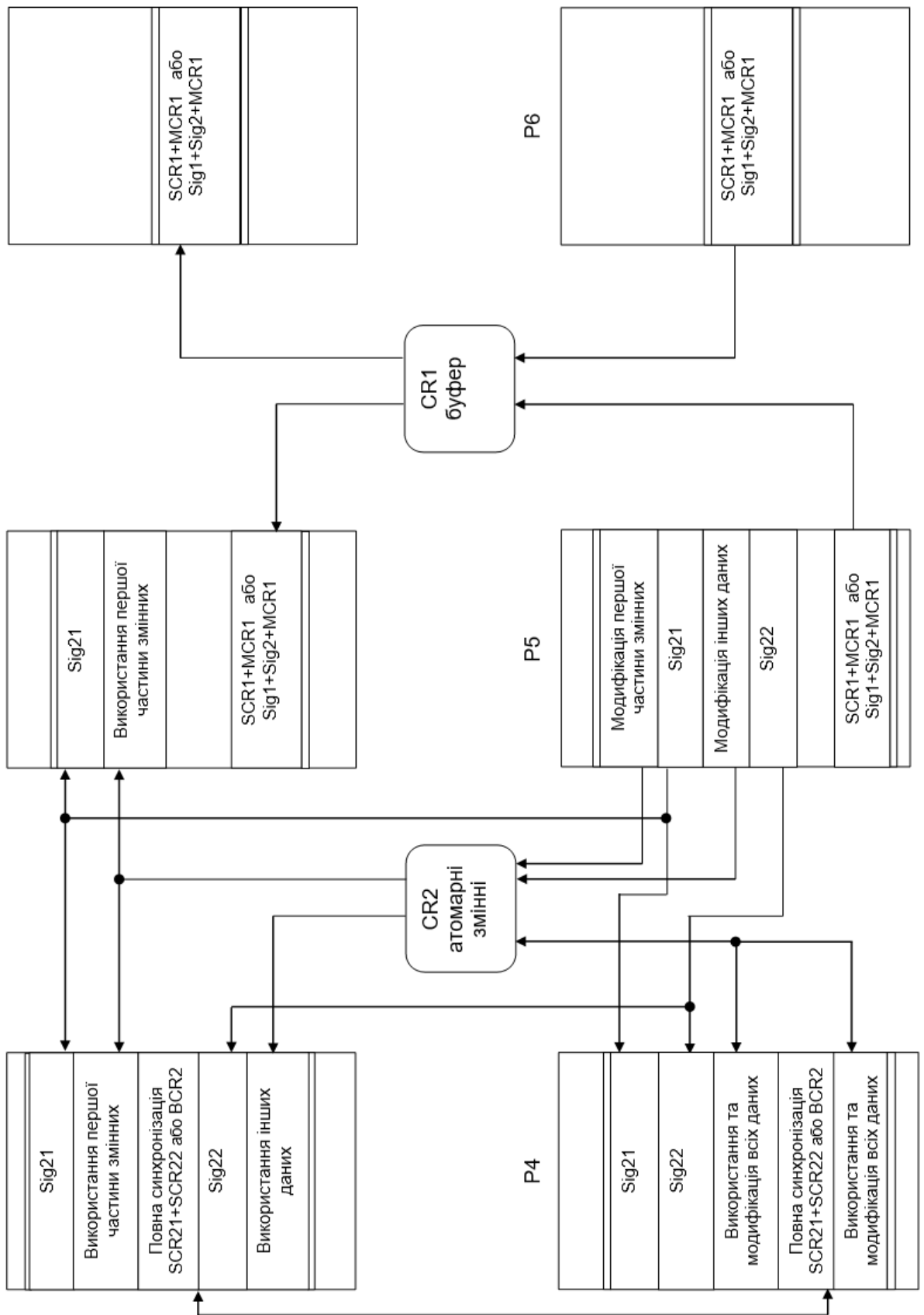


СХЕМА 5

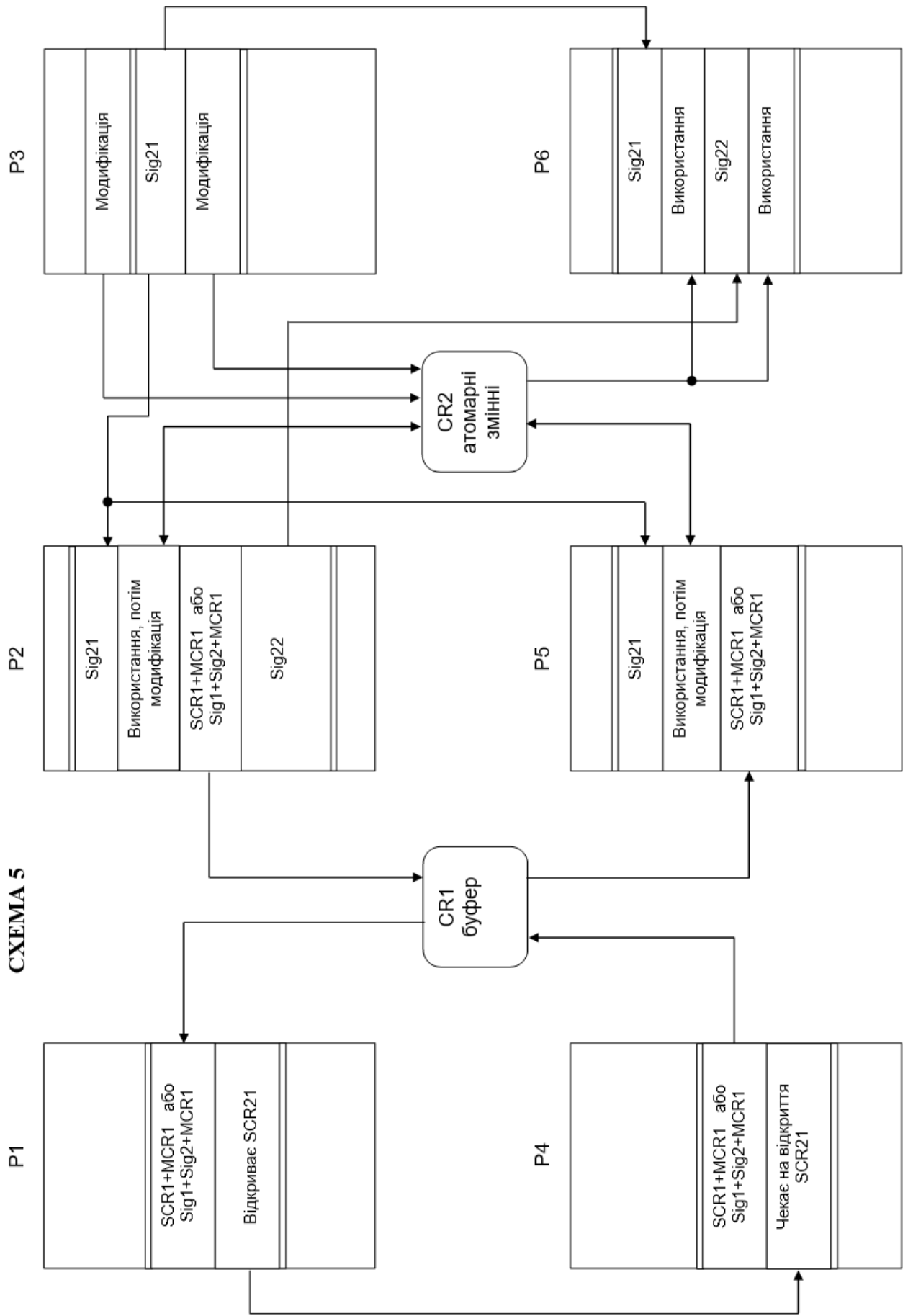


СХЕМА 6

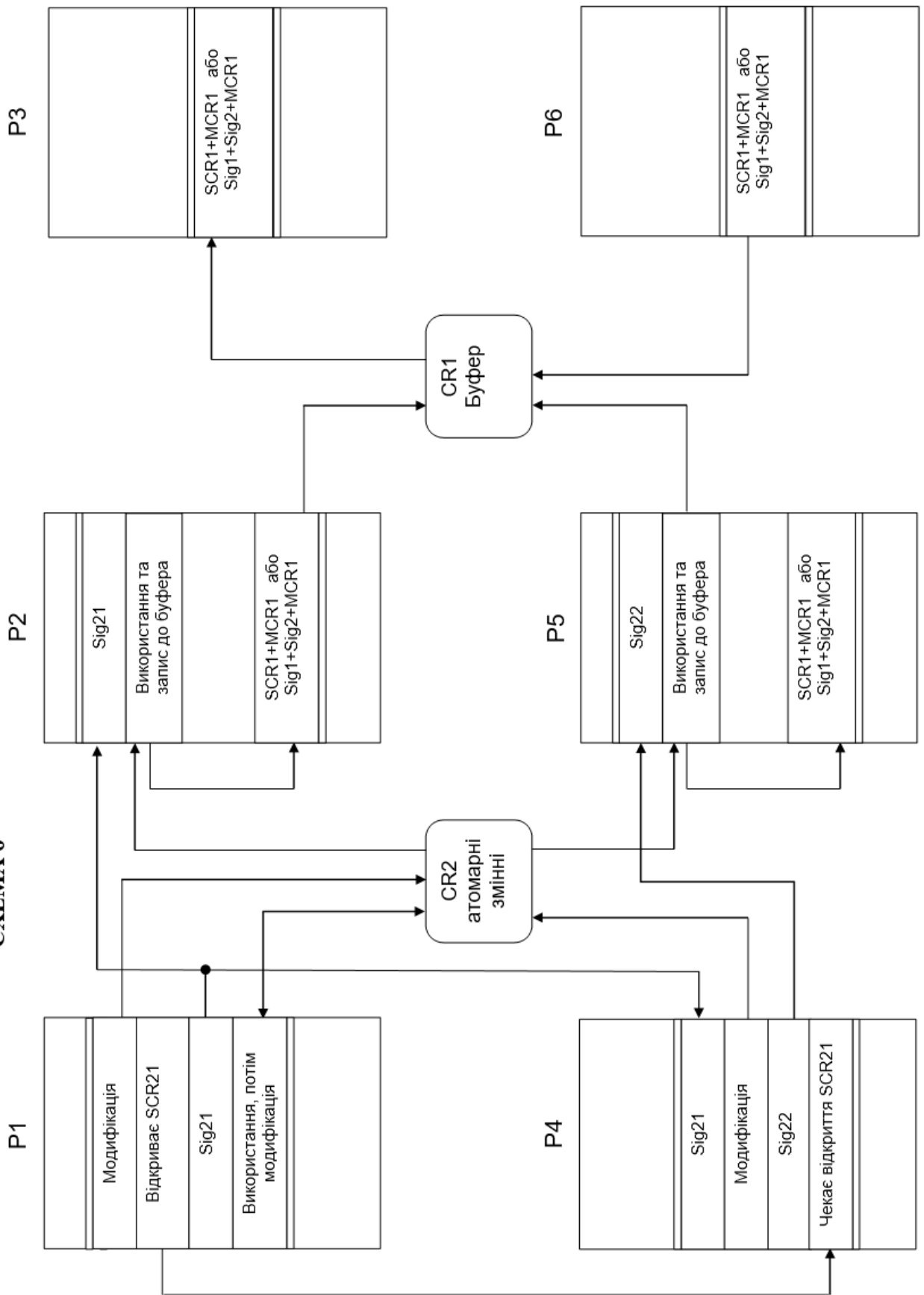


СХЕМА 7

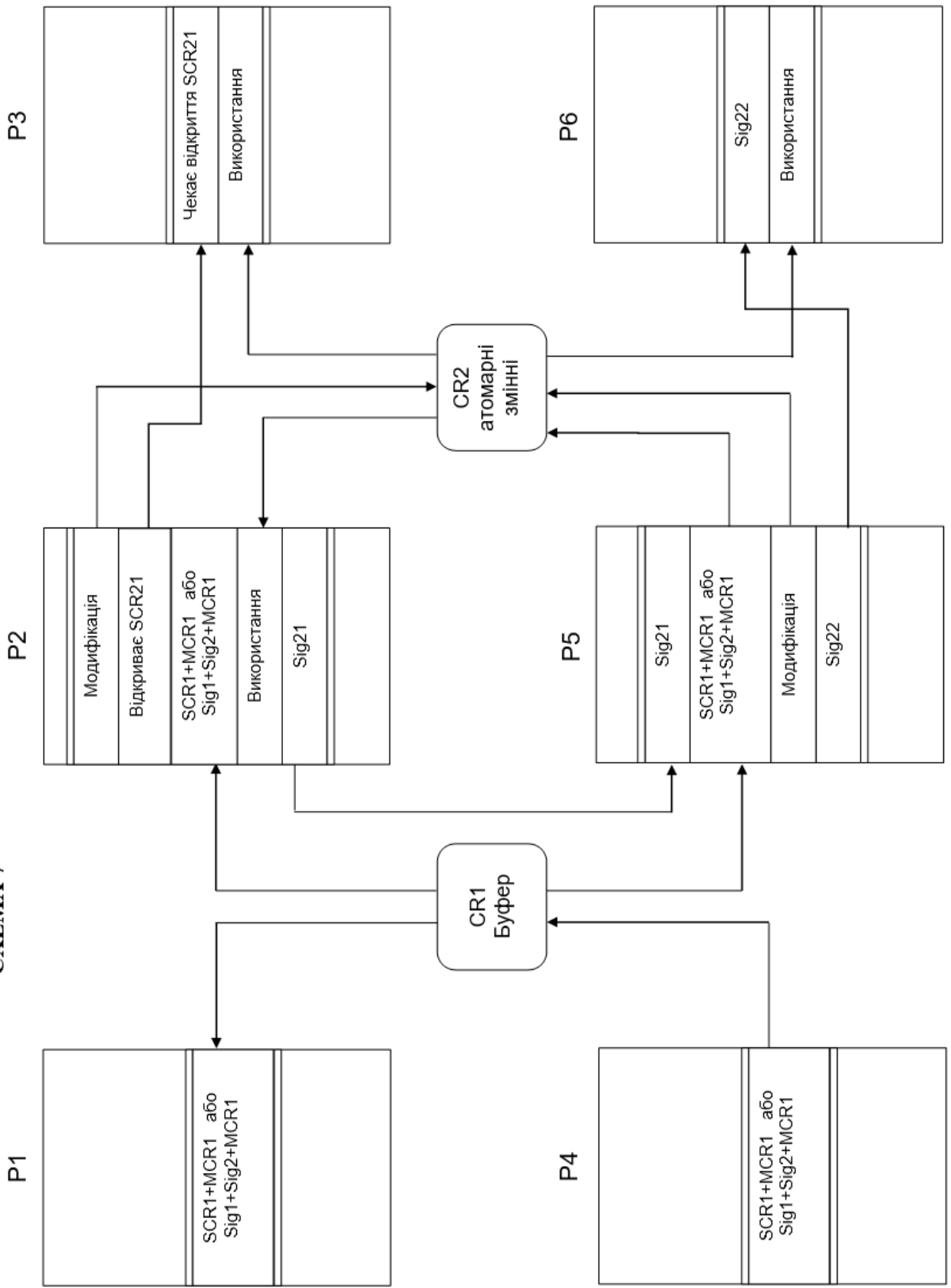


СХЕМА 8

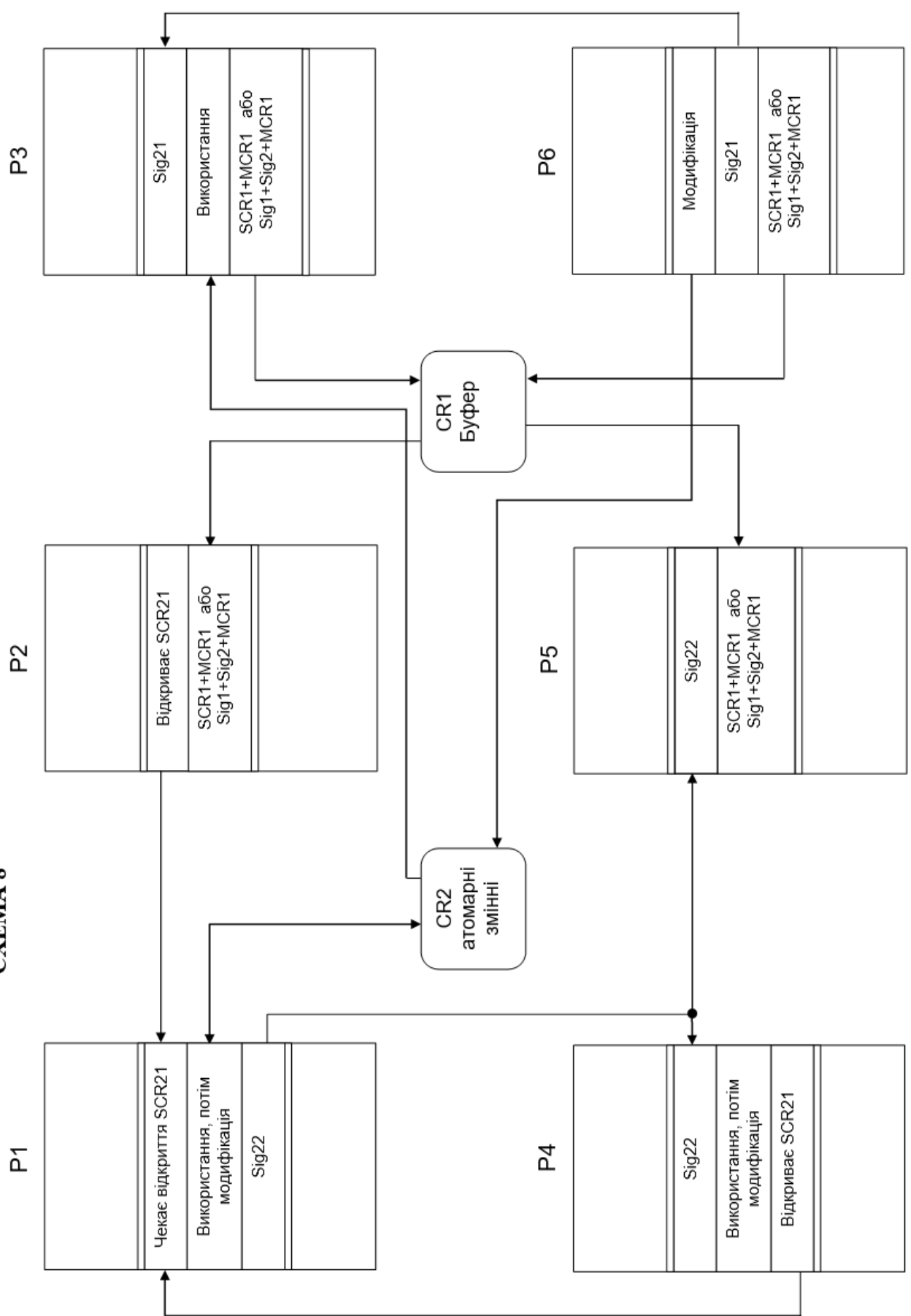
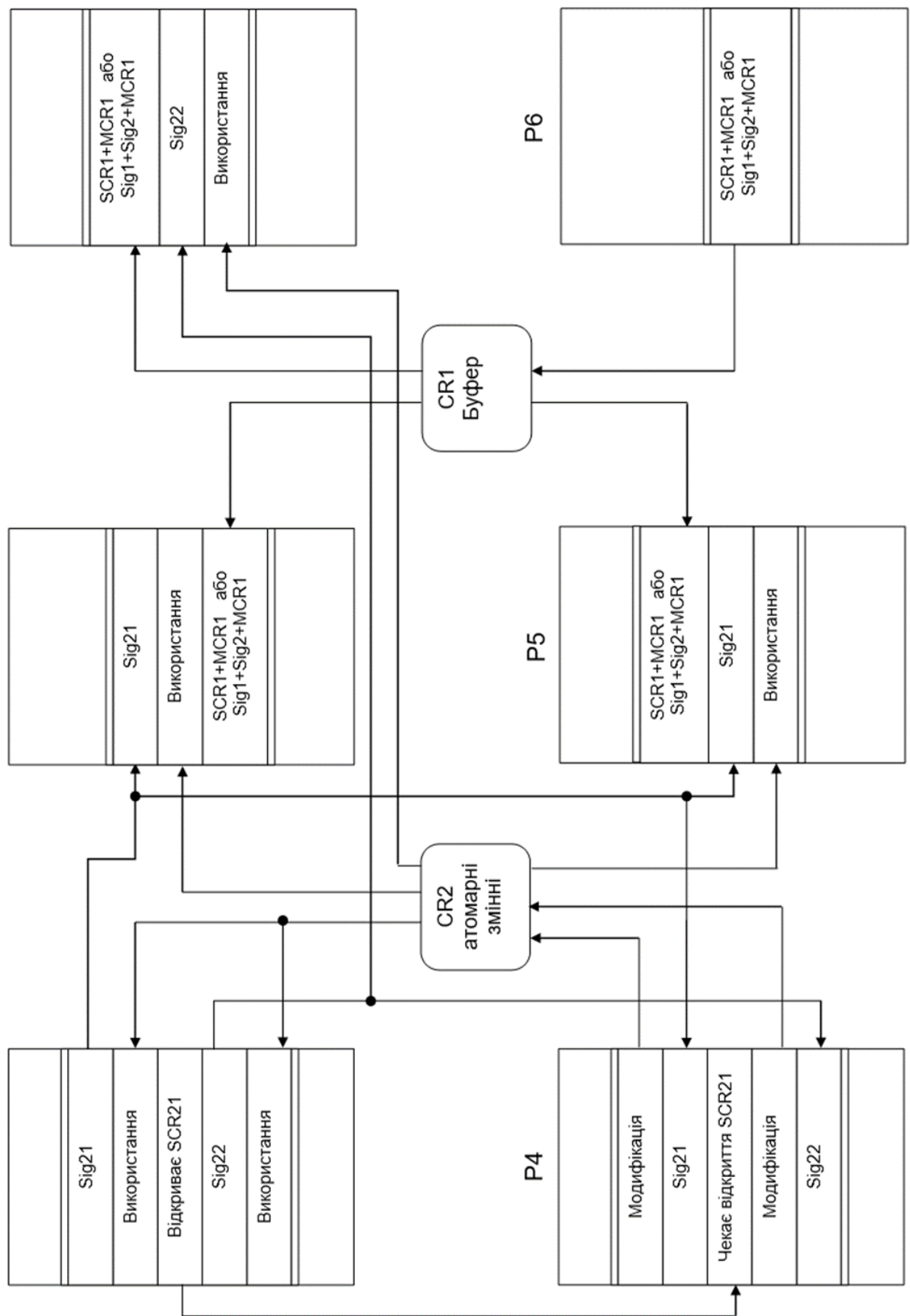


СХЕМА 10



ЛАБОРАТОРНА РОБОТА №5. СТВОРЕННЯ ПАРАЛЕЛЬНИХ ПОТОКІВ МОВИ JAVA ТА ЗАСОБИ ЇХ ВЗАЄМОДІЇ

Постановка задачі

1. Опрацювати всі надані лектором приклади коду паралельних потоків по темі «Багатопоточне програмування мовою Java», що знаходяться в директоріях **01_Creation**, **02_Threads_ID**, **03_Programs_to_Schemas_from_Conspect** та **04_Common_resource_Monitor** тобто:
 - вміти запускати всі ці приклади і отримувати результати на захисті лабораторної роботи;
 - знати які структури даних та конструкції взаємодії паралельних потоків описані в коді кожного прикладу та як вони працюють, а також вміти це пояснити на захисті лабораторної роботи;
 - розібратися з теоретичними ситуаціями, які відображують дані приклади, а також вміти їх розказати та пояснити на захисті лабораторної роботи;
2. Бути готовими до виконання модифікацій будь-яких з цих прикладів на захисті лабораторної роботи.

Контрольні питання

1. Напишіть команду компіляції файлу з багатопотоковою програмою та запустіть програму на виконання.
2. Що таке ідентифікатор потоку і як його можна отримати у програмі?
3. Дві головні проблеми при рішенні підзадачі синхронізації.
4. Завдання взаємного виключення: два підходи до його рішення.
5. В чому полягає підхід рішення завдання взаємного виключення, що ґрунтується на контролі процесів/потоків?
6. Завдання (проблема) несинхронної роботи без обміну інформації процесів/потоків.
7. Поняття семафора, види семафорів та алгоритми виконання операцій над семафорами.

8. Застосування двійкового семафору для рішення завдання взаємного виключення: схема, пояснення, описи даних та методи роботи із семафорами пакета **java.util.concurrent.Semaphore**.
9. Застосування двійкового семафору для рішення завдання простої (неповної) синхронізації: схема, пояснення та методи роботи із семафорами пакета **java.util.concurrent.Semaphore**.
10. Застосування двійкових семафорів для рішення завдання повної синхронізації: схема, пояснення та методи роботи із семафорами пакета **java.util.concurrent.Semaphore**.
11. Поняття м'ютекса та алгоритми виконання операцій над м'ютексом.
12. Застосування м'ютекса (**ReentrantLock**) для рішення завдання взаємного виключення: схема, пояснення, описи даних та методи роботи із м'ютексами пакета **java.util.concurrent.locks.***.
13. Поняття бар'єра та особливості використання бар'єрів пакета **java.util.concurrent.CyclicBarrier**.
14. Застосування бар'єра (**CyclicBarrier**) для рішення завдання повної синхронізації: схема, пояснення та методи роботи із бар'єрами пакета **java.util.concurrent.CyclicBarrier**.
15. В чому полягає підхід рішення завдання взаємного виключення, що ґрунтується на контролі спільного ресурсу?
16. Поняття монітора та його відмінності від інших засобів взаємодії паралельних потоків.
17. Особливості реалізації конструкції монітора в мові Java та використання методів **wait()** і **notify()** для організації коректного використання спільного ресурсу роботи при його захисті за допомогою монітора (на основі прикладів папки **04_Common_resource_Monitor**).

ЛАБОРАТОРНА РОБОТА №6. КОМПЛЕКСНЕ ВИКОРИСТАННЯ ЗАСОБІВ ВЗАЄМОДІЇ ПАРАЛЕЛЬНИХ ПОТОКІВ МОВИ JAVA

Постановка завдання

1. Написати програму на мові Java, яка реалізує роботу паралельних потоків згідно заданої за варіантом схеми. Особливості реалізації синхронізації паралельних потоків та взаємного виключення потоків при доступі до спільних ресурсів задані за варіантами у таблиці 1.
2. При написанні програми виконати повне трасування роботи програми за допомогою операторів друку, тобто розставити в програмі оператори друку таким чином, щоб можна було прослідкувати всі варіанти виконання паралельних потоків і впевнитись у коректності роботи програми.
3. Всі потоки повинні бути реалізовані у вигляді нескінченних циклів.
4. Оскільки синхронізація за допомогою семафорів SCR21, SCR22 згідно завдання розташована всередині нескінченних циклів, то відразу після виконання синхронізації ці семафори повинні бути знову встановлені у початковий закритий стан.
5. Закінчення програми можна виконати двома способами:
 - примусовим перериванням за допомогою натиснення комбінації клавіш Ctrl+C;
 - оператором break при виконанні умови, яка стає істинною, коли буфер спільного ресурсу повністю заповнюється і повністю звільняється мінімум по два рази.
6. Якщо при реалізації паралельних потоків був використаний метод sleep(), то передбачити режим запуску програми з «відключеними» викликами sleep().
7. Виконати налагодження написаної програми.

Зміст звіту

1. Загальна постановка завдання.
2. Завдання конкретного варіанту.
3. Схема паралельних потоків свого варіанту.
4. Текст програми.
5. Декілька протоколів роботи програми, які демонструють різні випадки роботи паралельних потоків.

Контрольні питання

1. Вміти відповідати на будь-які питання стосовно задачі «Producer-Consumer» та всіх структур даних, що використовуються в програмі (стек у вигляді вектора, циклічний буфер тощо).
2. Знати принципи організації та засоби взаємодії паралельних потоків із загальнотеоретичної точки зору згідно теоретичній частині лекційного матеріалу, а також реалізацію цих засобів у мові Java щонайменше на прикладах, наданих до лекційного матеріалу.

Варіанти завдань

Таблиця 1

№ варіанту	№ схеми	Спільний ресурс 1 CR1 (буфер обміну даними)		Спільний ресурс 2 CR2	Засоби синхронізації потоків
		Структура даних, що використовується у якості спільного ресурсу 1	Засіб взаємного виключення при доступі до спільного ресурсу 1	Засіб взаємного виключення при доступі до змінних спільного ресурсу 2	
1	1	Стек у вигляді вектора	Монітор	М'ютекс	Задані безпосередньо на схемі
2	2	Циклічний буфер у вигляді вектора	Монітор	М'ютекс	Задані безпосередньо на схемі
3	3	Циклічний буфер у вигляді вектора	Монітор	М'ютекс	Задані безпосередньо на схемі
4	4	Стек у вигляді вектора	Монітор	М'ютекс	Задані безпосередньо на схемі
5	5	Циклічний буфер у вигляді вектора	Монітор	М'ютекс	Задані безпосередньо на схемі
6	6	Стек у вигляді вектора	Монітор	М'ютекс	Задані безпосередньо на схемі
7	7	Стек у вигляді вектора	Монітор	М'ютекс	Задані безпосередньо на схемі
8	8	Циклічний буфер у вигляді вектора	Монітор	М'ютекс	Задані безпосередньо на схемі
9	9	Циклічний буфер у вигляді вектора	Монітор	М'ютекс	Задані безпосередньо на схемі
10	10	Стек у вигляді вектора	Монітор	М'ютекс	Задані безпосередньо на схемі
11	11	Циклічний буфер у вигляді вектора	Монітор	М'ютекс	Задані безпосередньо на схемі
12	12	Циклічний буфер у вигляді вектора	Монітор	М'ютекс	Задані безпосередньо на схемі
13	13	Стек у вигляді вектора	Монітор	М'ютекс	Задані безпосередньо на схемі
14	14	Циклічний буфер у вигляді вектора	Монітор	М'ютекс	Задані безпосередньо на схемі
15	15	Стек у вигляді вектора	Монітор	М'ютекс	Задані безпосередньо на схемі

16	16	Стек у вигляді вектора	Монітор	М'ютекс	Задані безпосередньо на схемі
17	17	Циклічний буфер у вигляді вектора	Монітор	М'ютекс	Задані безпосередньо на схемі
18	18	Стек у вигляді вектора	Монітор	М'ютекс	Задані безпосередньо на схемі
19	19	Стек у вигляді вектора	Монітор	М'ютекс	Задані безпосередньо на схемі
20	20	Циклічний буфер у вигляді вектора	Монітор	М'ютекс	Задані безпосередньо на схемі
21	21	Циклічний буфер у вигляді вектора	Монітор	М'ютекс	Задані безпосередньо на схемі
22	22	Стек у вигляді вектора	Монітор	М'ютекс	Задані безпосередньо на схемі
23	23	Стек у вигляді вектора	Монітор	М'ютекс	Задані безпосередньо на схемі
24	24	Циклічний буфер у вигляді вектора	Монітор	М'ютекс	Задані безпосередньо на схемі
25	25	Циклічний буфер у вигляді вектора	Монітор	М'ютекс	Задані безпосередньо на схемі
26	26	Стек у вигляді вектора	Монітор	М'ютекс	Задані безпосередньо на схемі
27	27	Стек у вигляді вектора	Монітор	М'ютекс	Задані безпосередньо на схемі
28	28	Циклічний буфер у вигляді вектора	Монітор	М'ютекс	Задані безпосередньо на схемі
29	29	Стек у вигляді вектора	Монітор	М'ютекс	Задані безпосередньо на схемі
30	30	Циклічний буфер у вигляді вектора	Монітор	М'ютекс	Задані безпосередньо на схемі

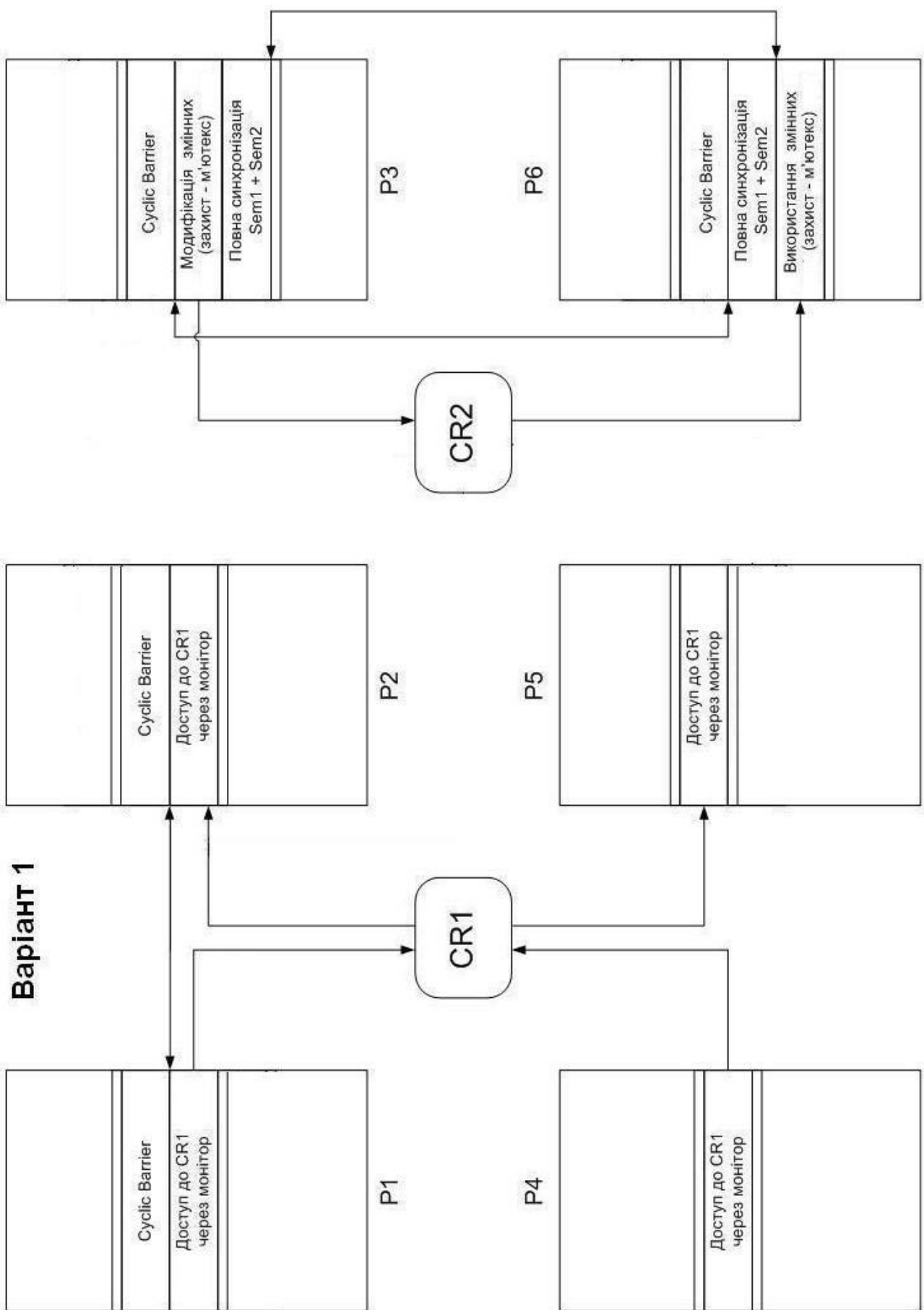
Схеми паралельних потоків за варіантами

Умовні позначення на схемах:

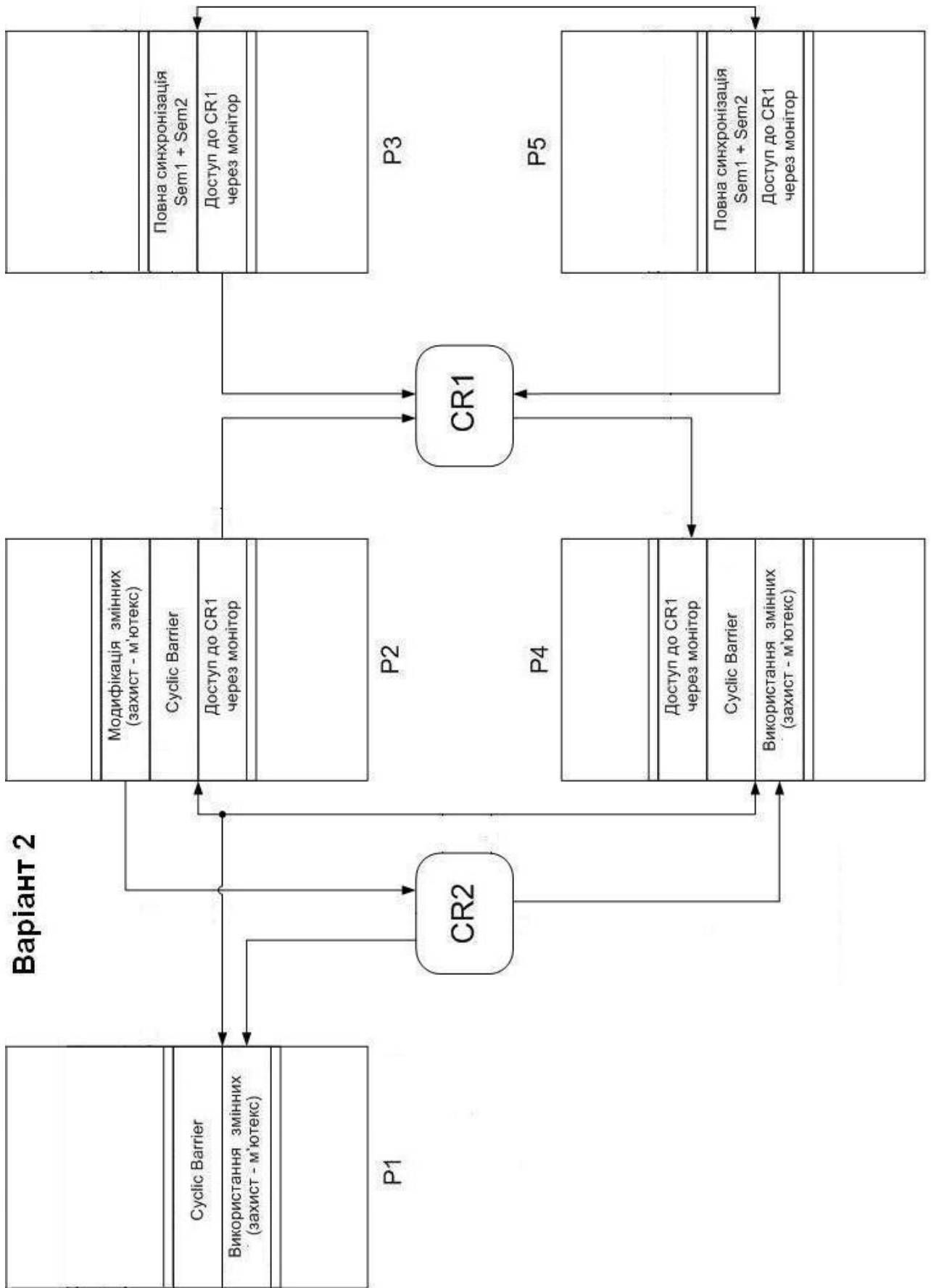
1. **Дві подвійні лінії (верхня та нижня)** означають, що дії між цими лініями повторюються циклічно. Тобто подвійні лінії використовуються аналогічно їх використанню в діаграмах дій.
2. **CR1** – спільний ресурс (common resource) у вигляді буфера для обміну даними між потоками-постачальниками і потоками-споживачами. В якості спільного ресурсу CR1 (буфера обміну даними) використати звичайний одномірний масив (вектор), а не колекції мови Java. Спосіб реалізації буфера визначається у таблиці 1 за варіантами.
3. **CR2** – спільний ресурс (common resource), який складається з неатомарних змінних (по одній змінній кожного з елементарних типів мови Java).
4. **Доступ до CR1 через монітор** – в точках з таким позначенням потрібно виконати доступ до спільного ресурсу CR1 за допомогою синхронізованих

методів монітора, вбудованого у клас цього спільного ресурсу. Вид доступу (запис чи читання) визначається напрямом відповідних стрілок.

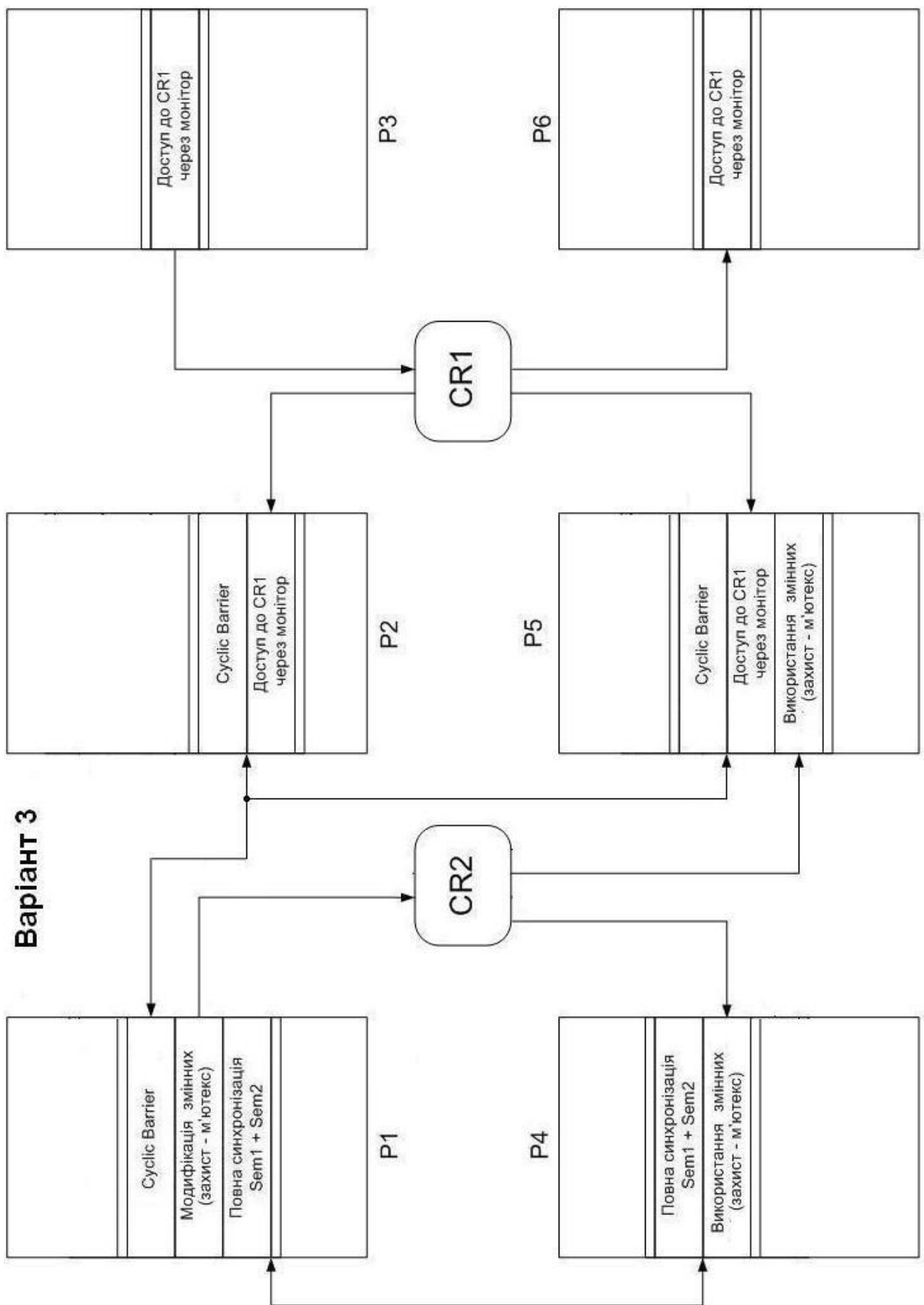
5. **Модифікація змінних (захист м'ютекс)** – в точках з таким позначенням потрібно виконати модифікацію значень неатомарних змінних спільного ресурсу CR2 значеннями відповідних типів. Ініціалізацію цих змінних виконати у тому ж потоці перед циклом. Взаємне виключення потоків від одночасного доступу до цих змінних виконати за допомогою м'ютекса.
6. **Використання змінних (захист м'ютекс)** – в точках з таким позначенням потрібно виконати читання значень неатомарних змінних спільного ресурсу CR2 та виконання над ними довільних дій. Взаємне виключення потоків від одночасного доступу до цих змінних виконати за допомогою м'ютекса.
7. **Повна синхронізація Sem1 + Sem2** – в точках з таким позначенням потрібно виконати повну синхронізацію двох потоків за допомогою двох двійкових семафорів Sem1 та Sem2.
8. **Cyclic Barrier** – в точках з таким позначенням потрібно виконати повну синхронізацію двох (чи трьох) потоків за допомогою засобу синхронізації типу «циклічний бар'єр».



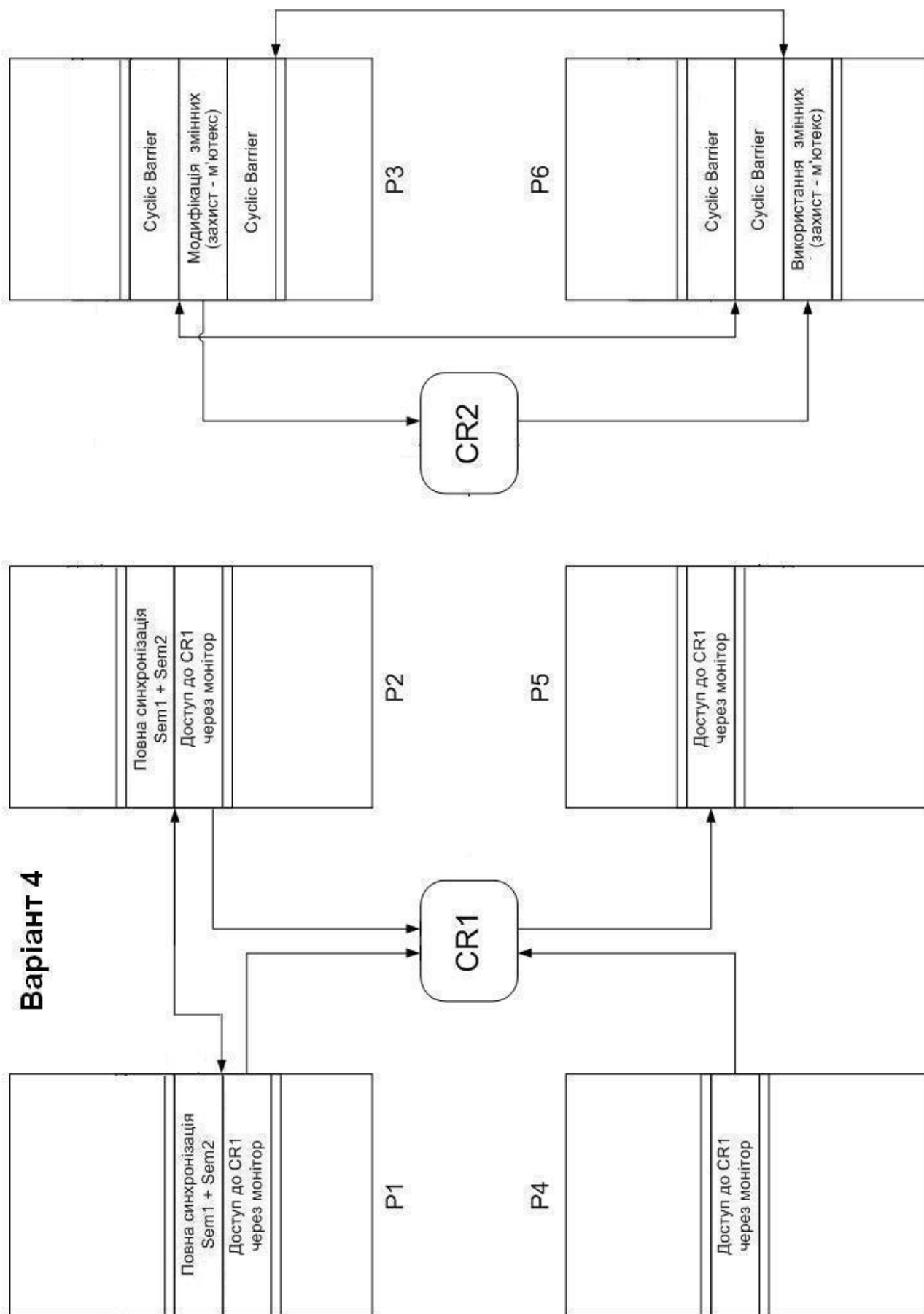
Варіант 2



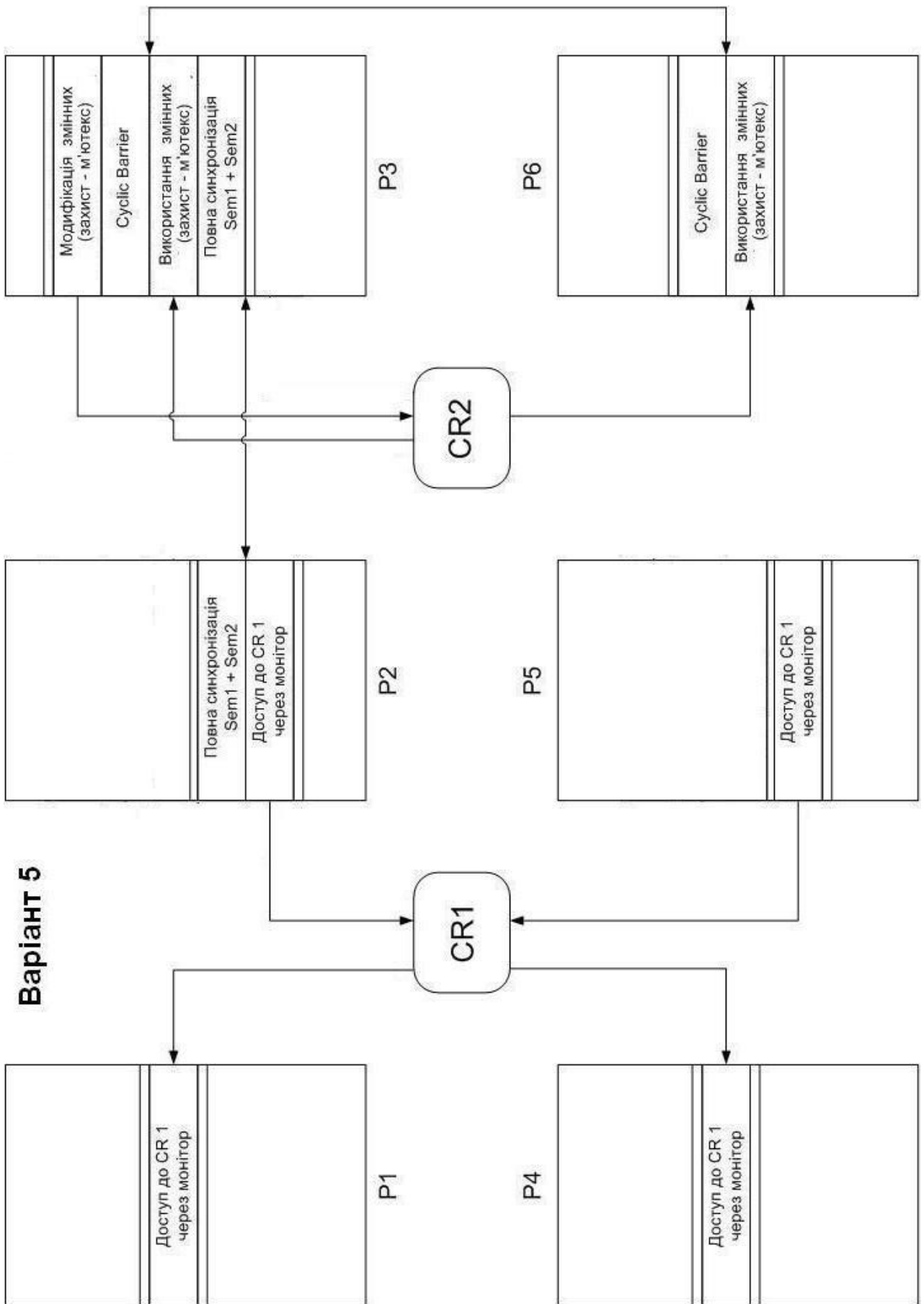
Варіант 3



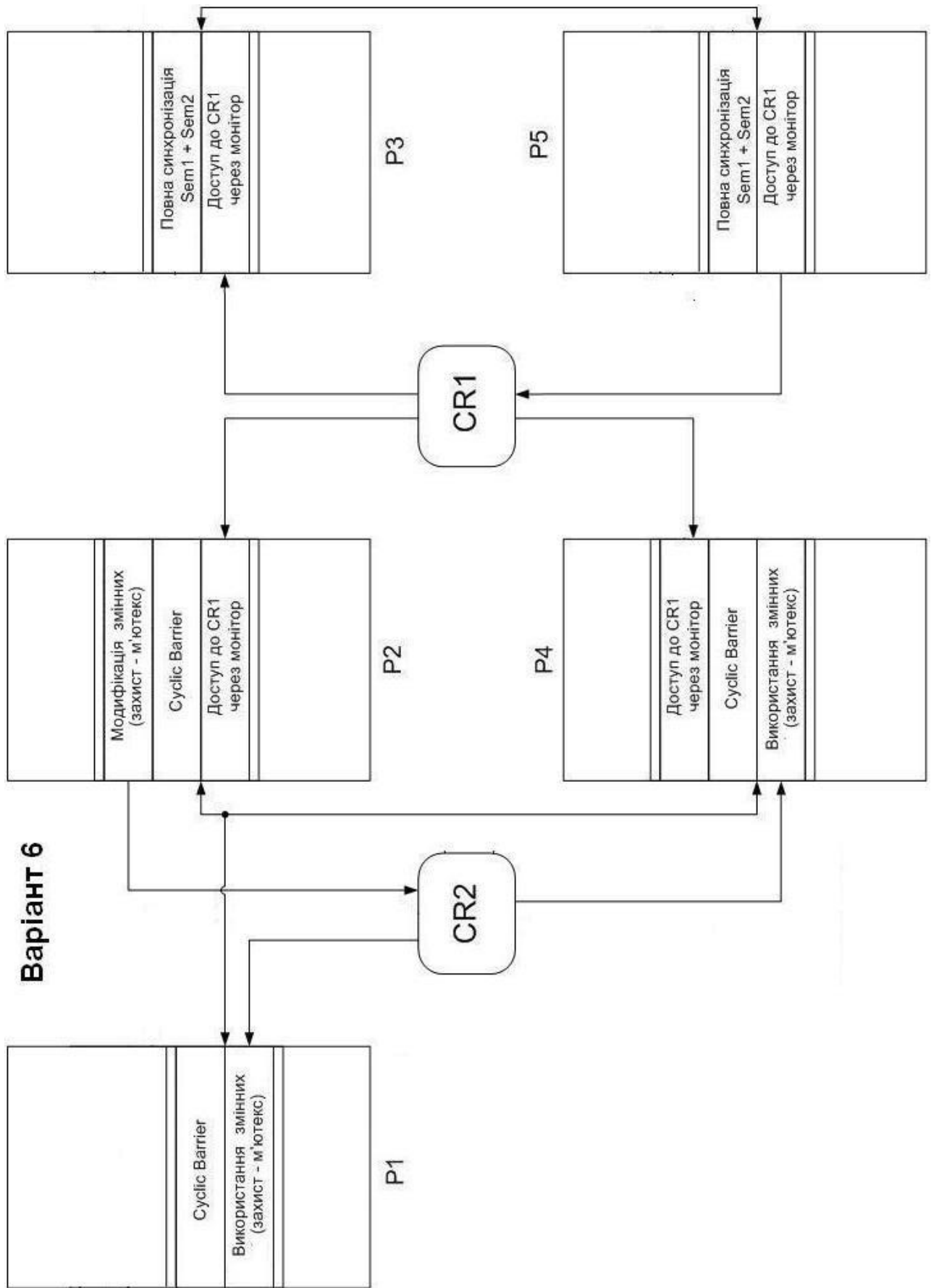
Варіант 4

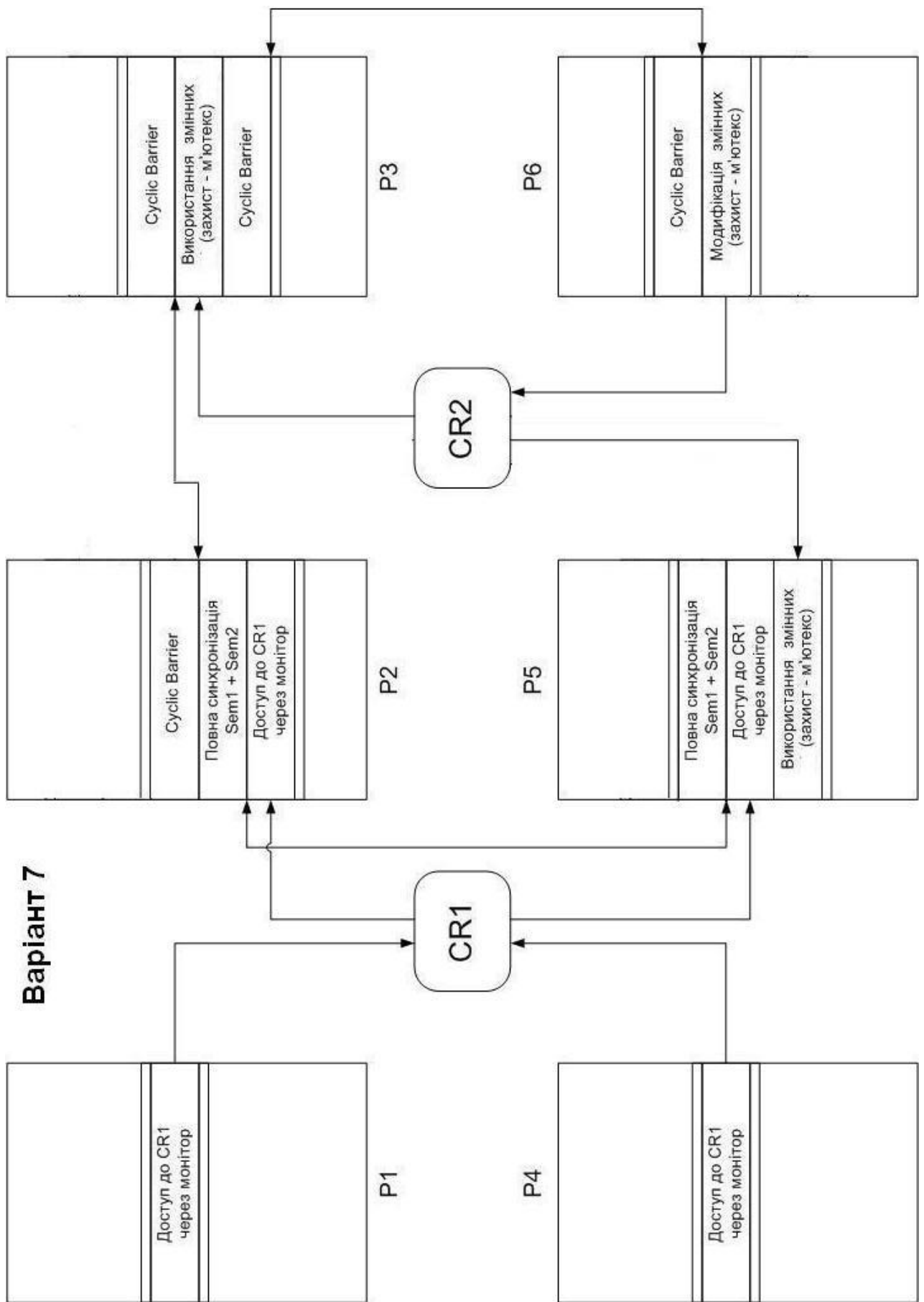


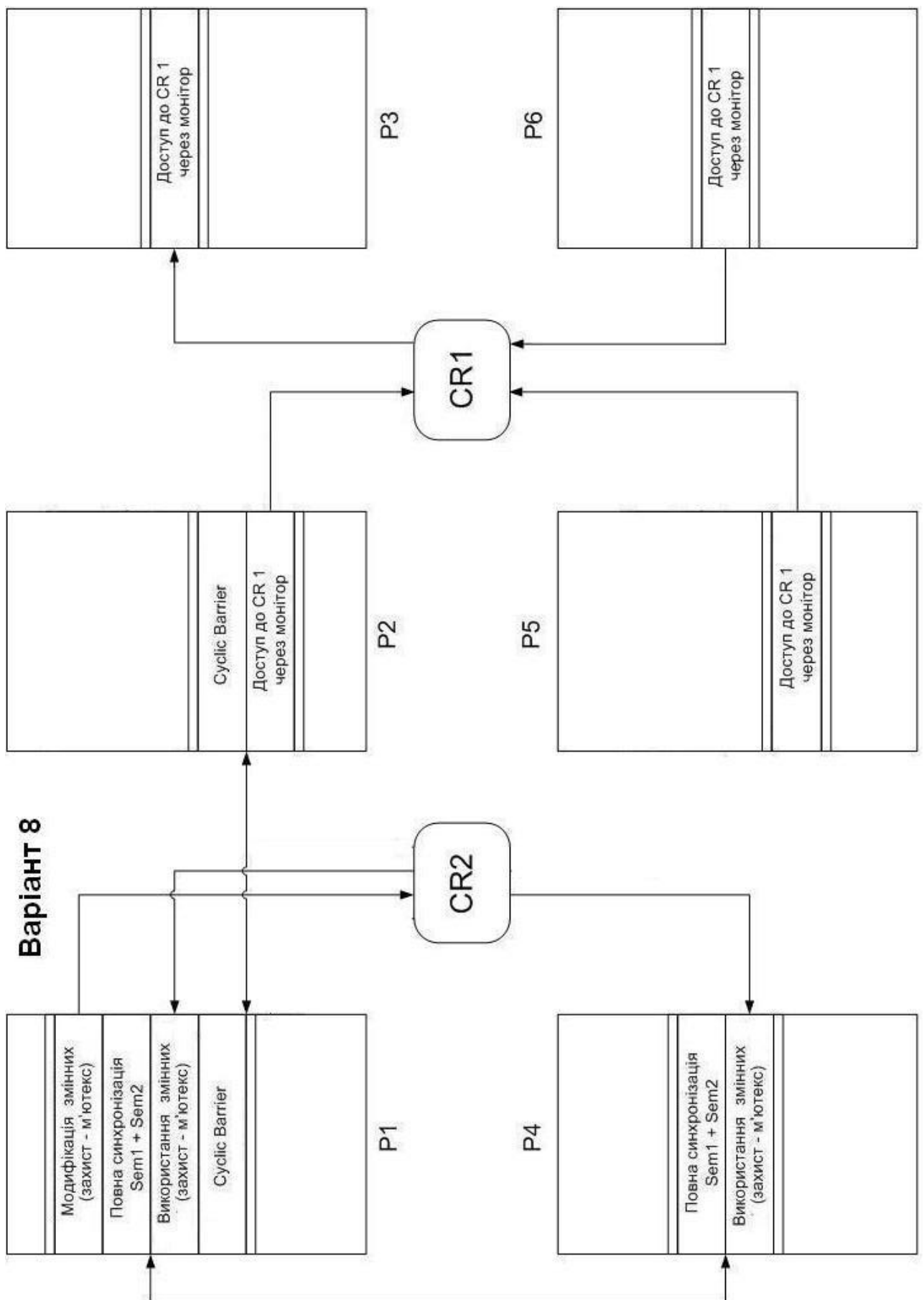
Варіант 5

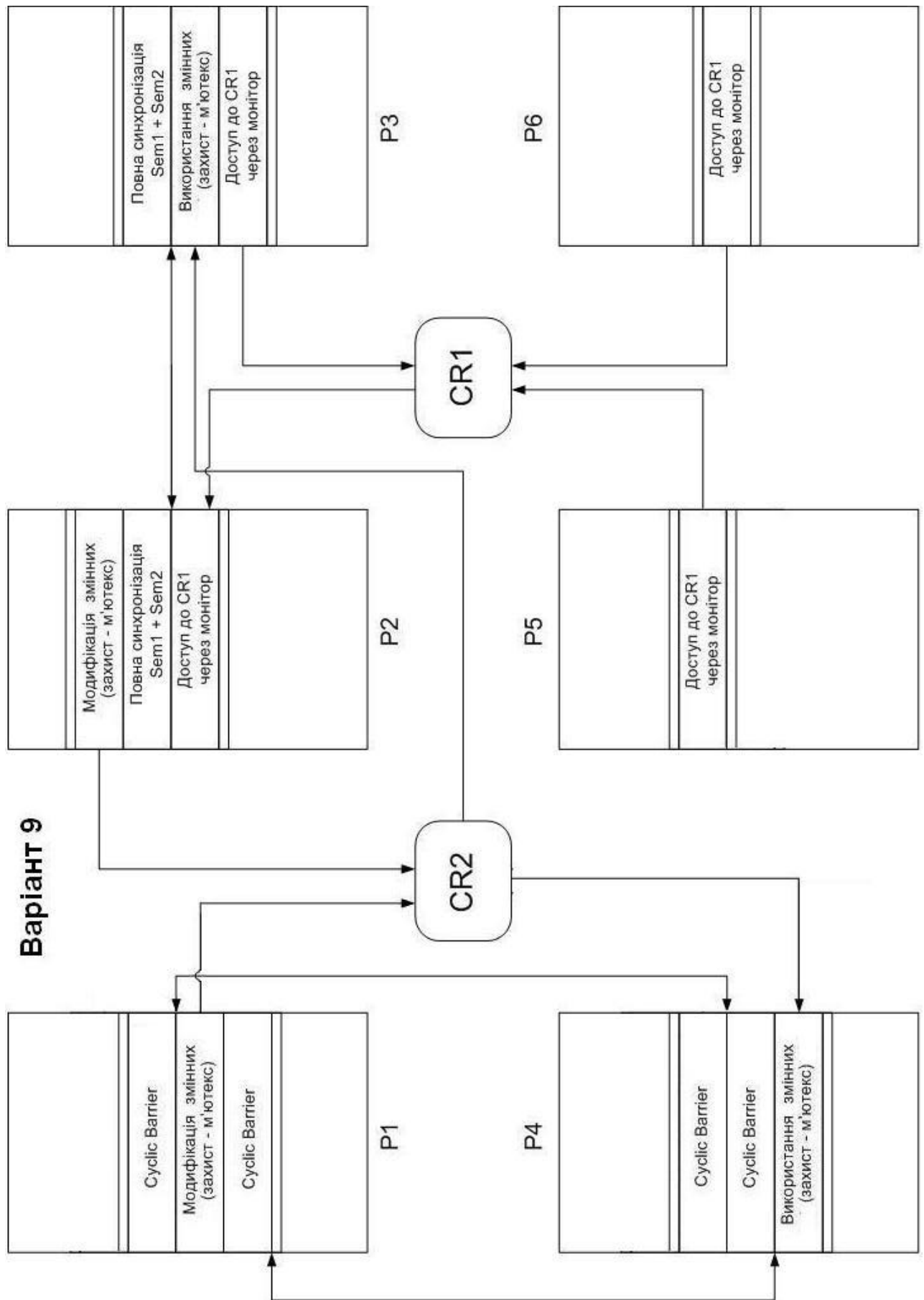


Варіант 6

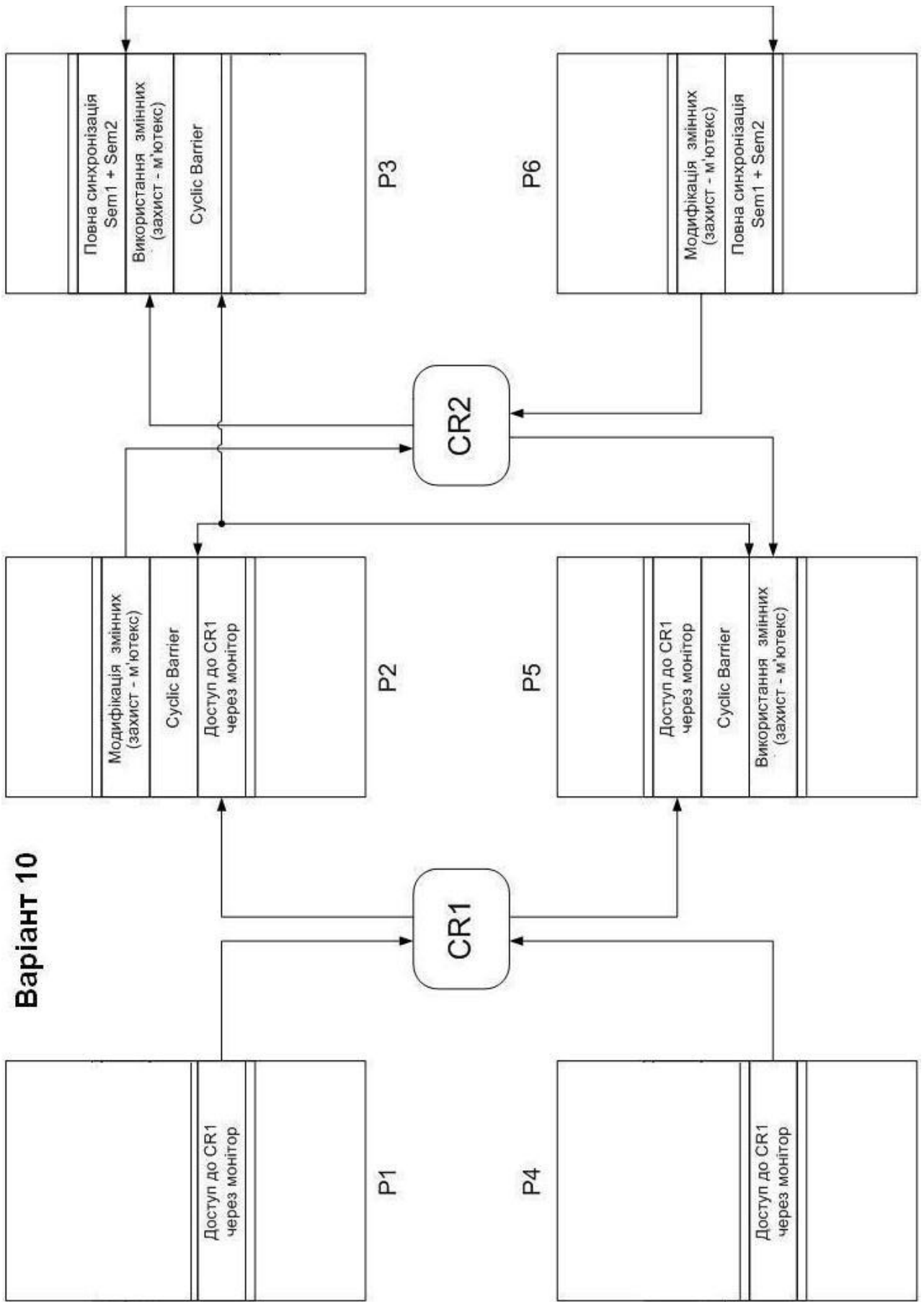


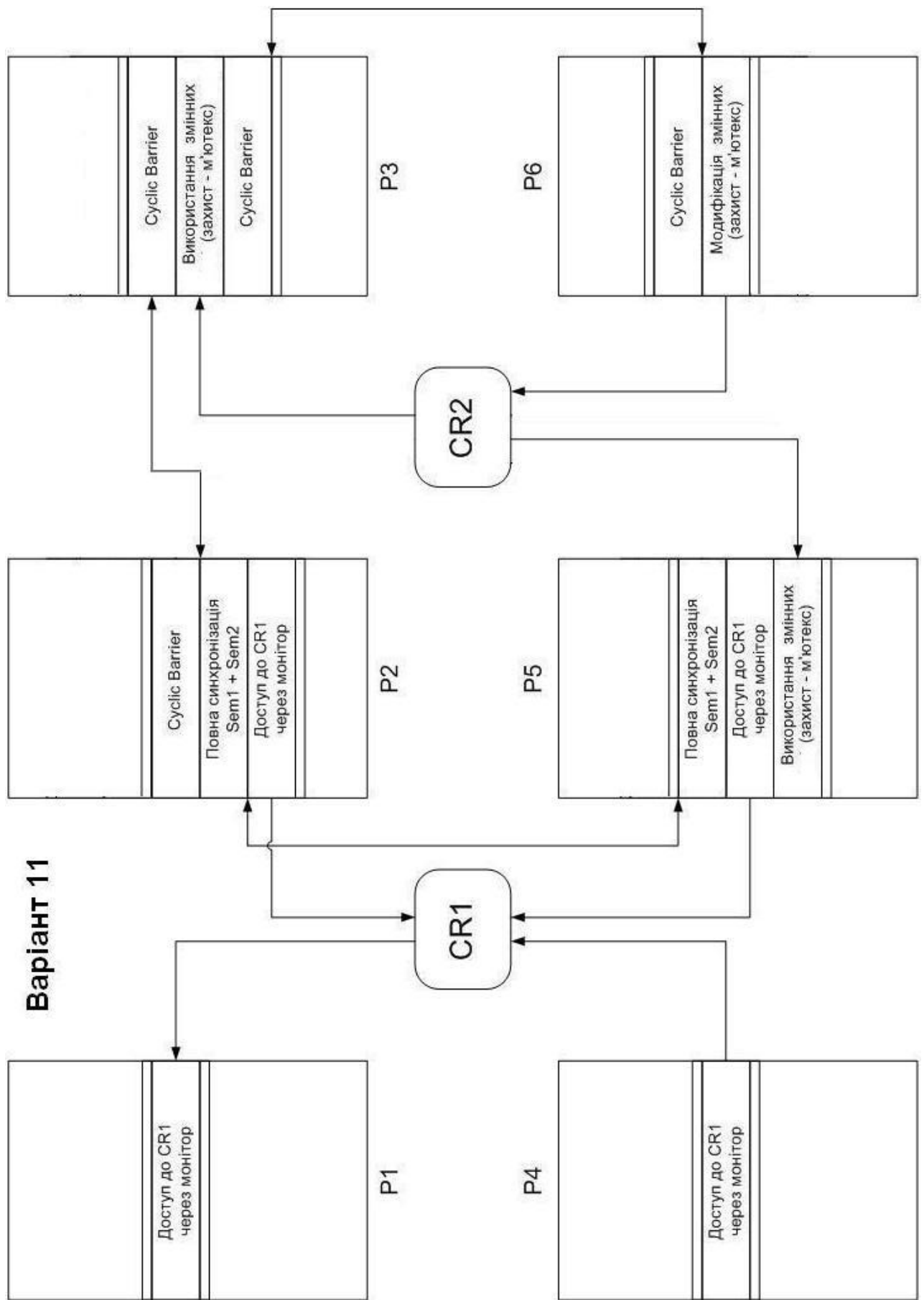




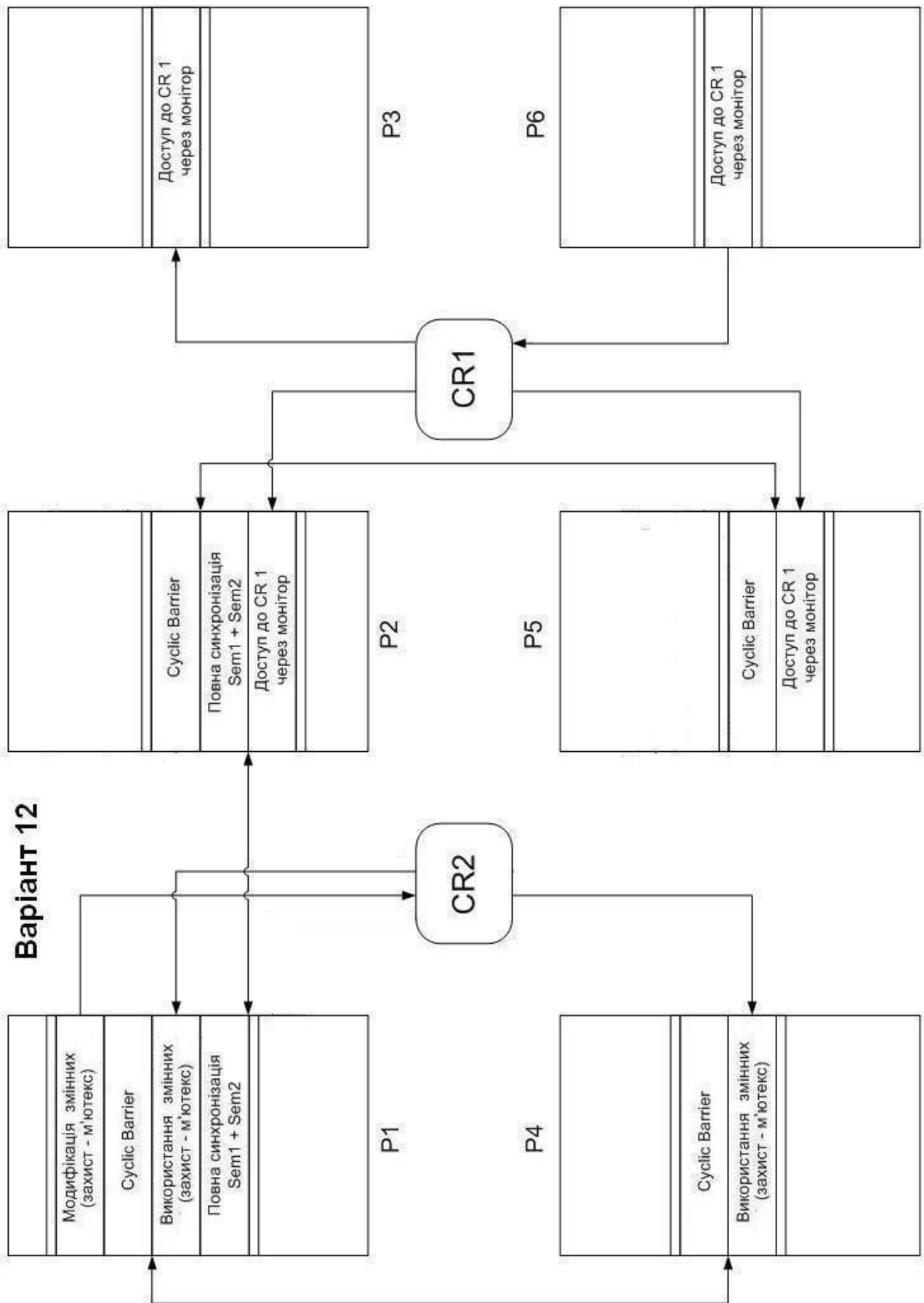


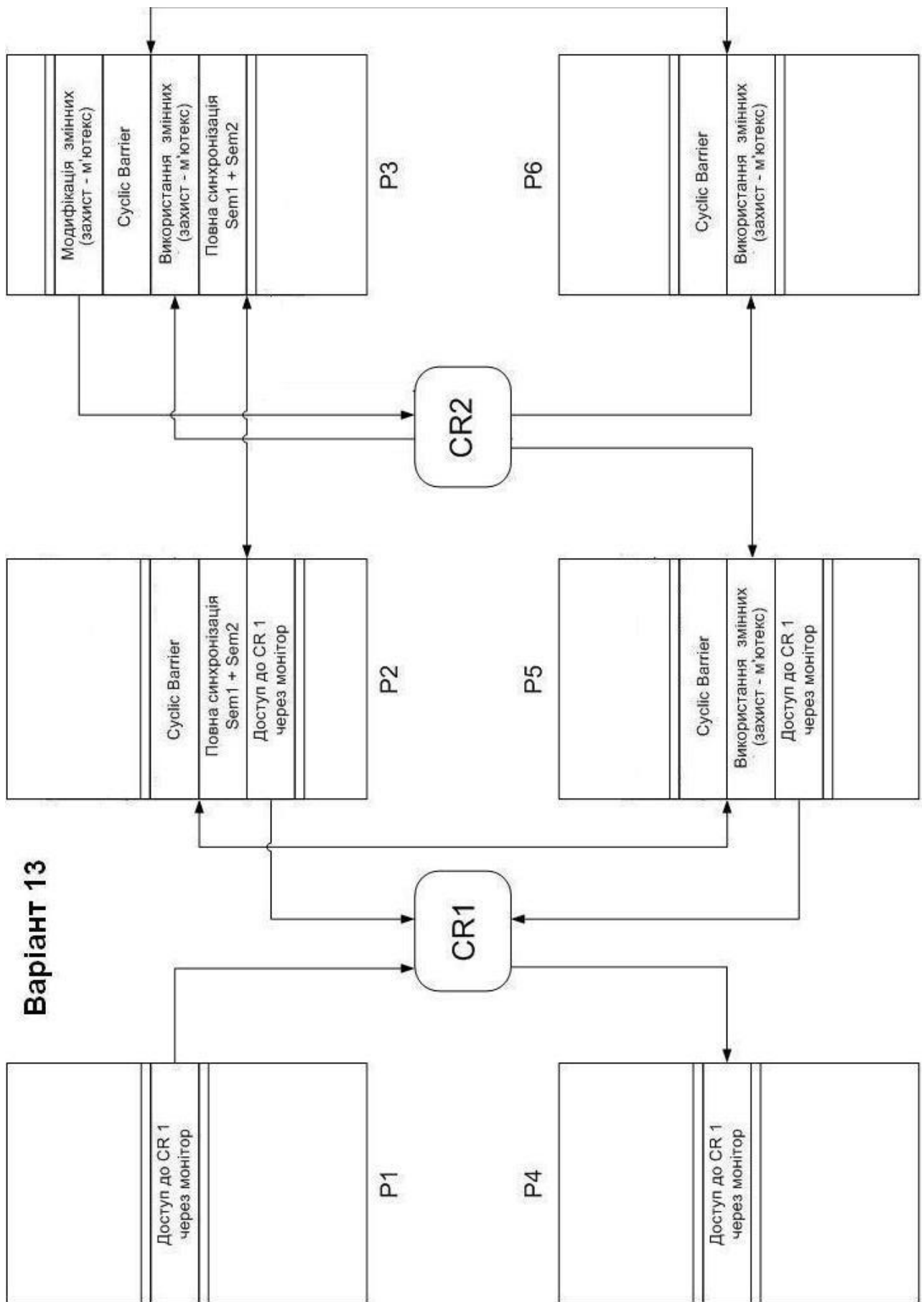
Варіант 10



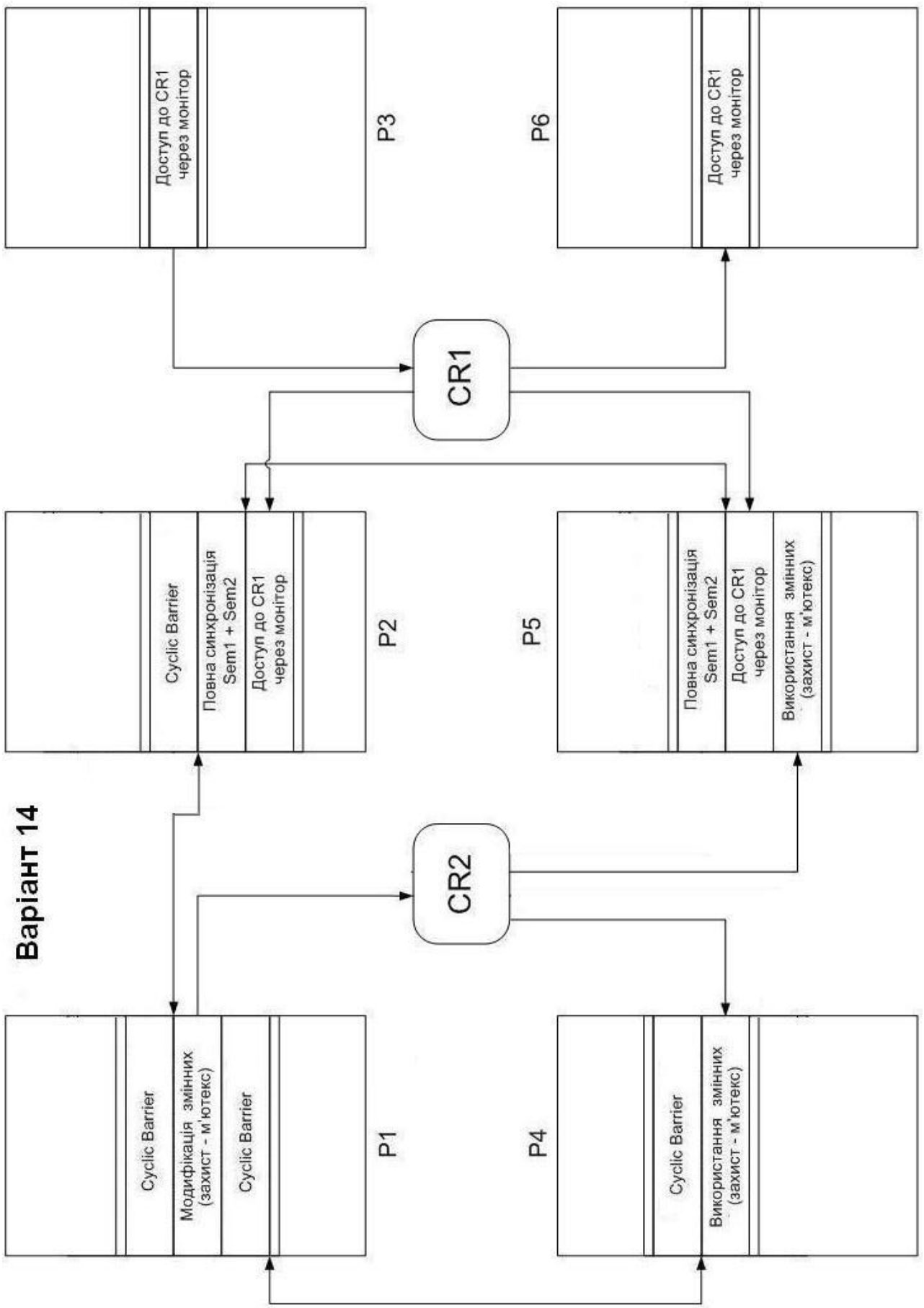


Варіант 12

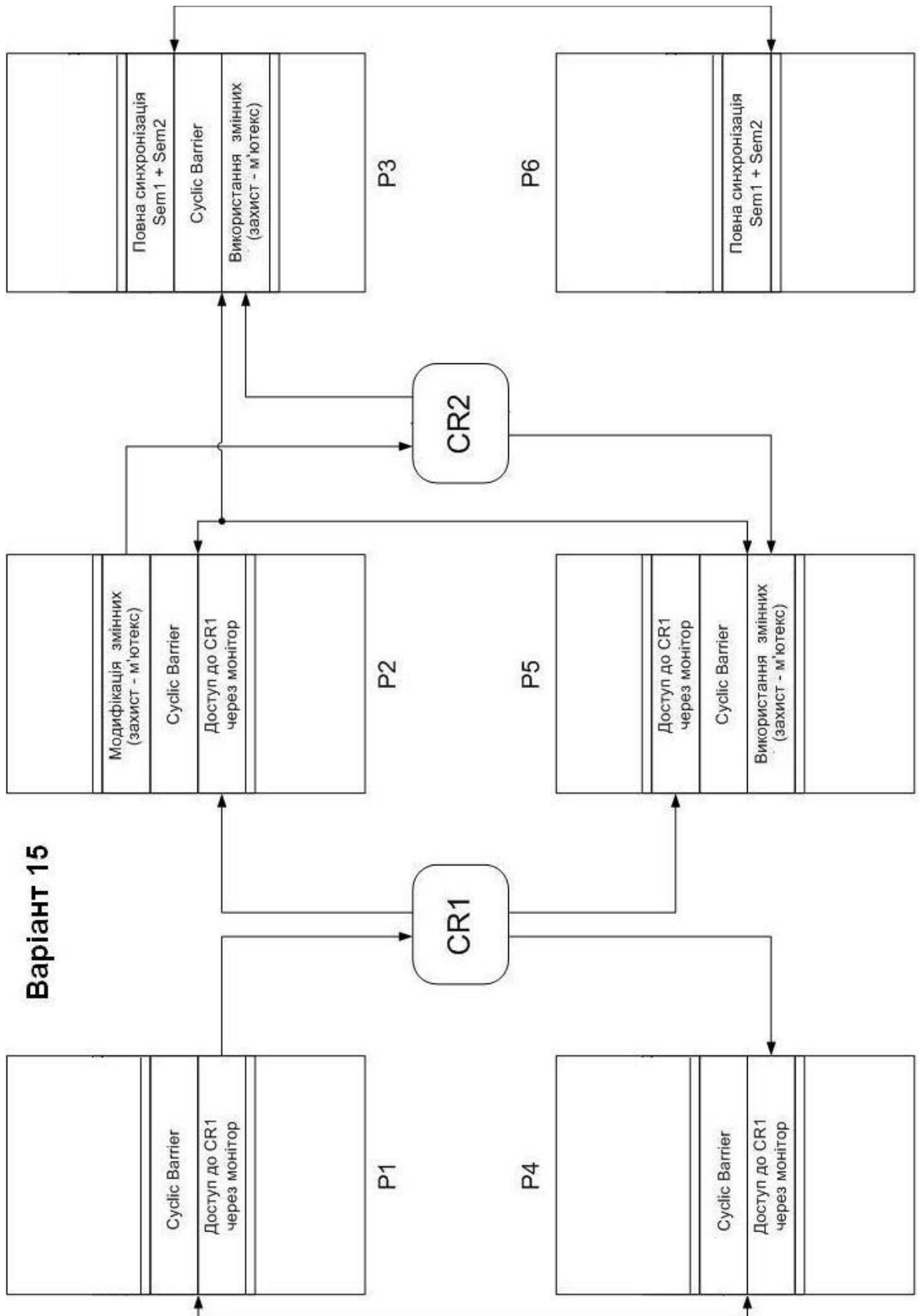


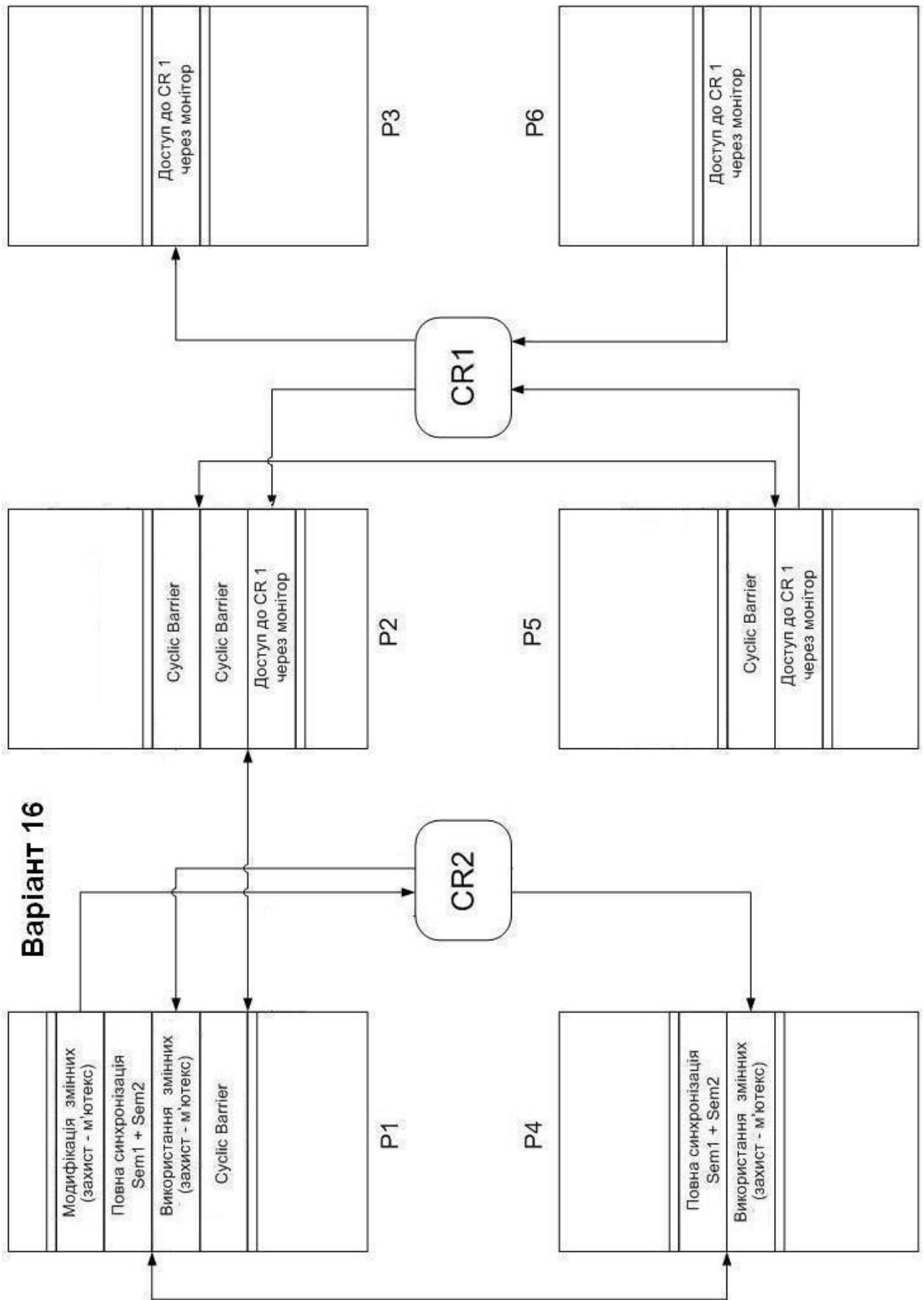


Варіант 14

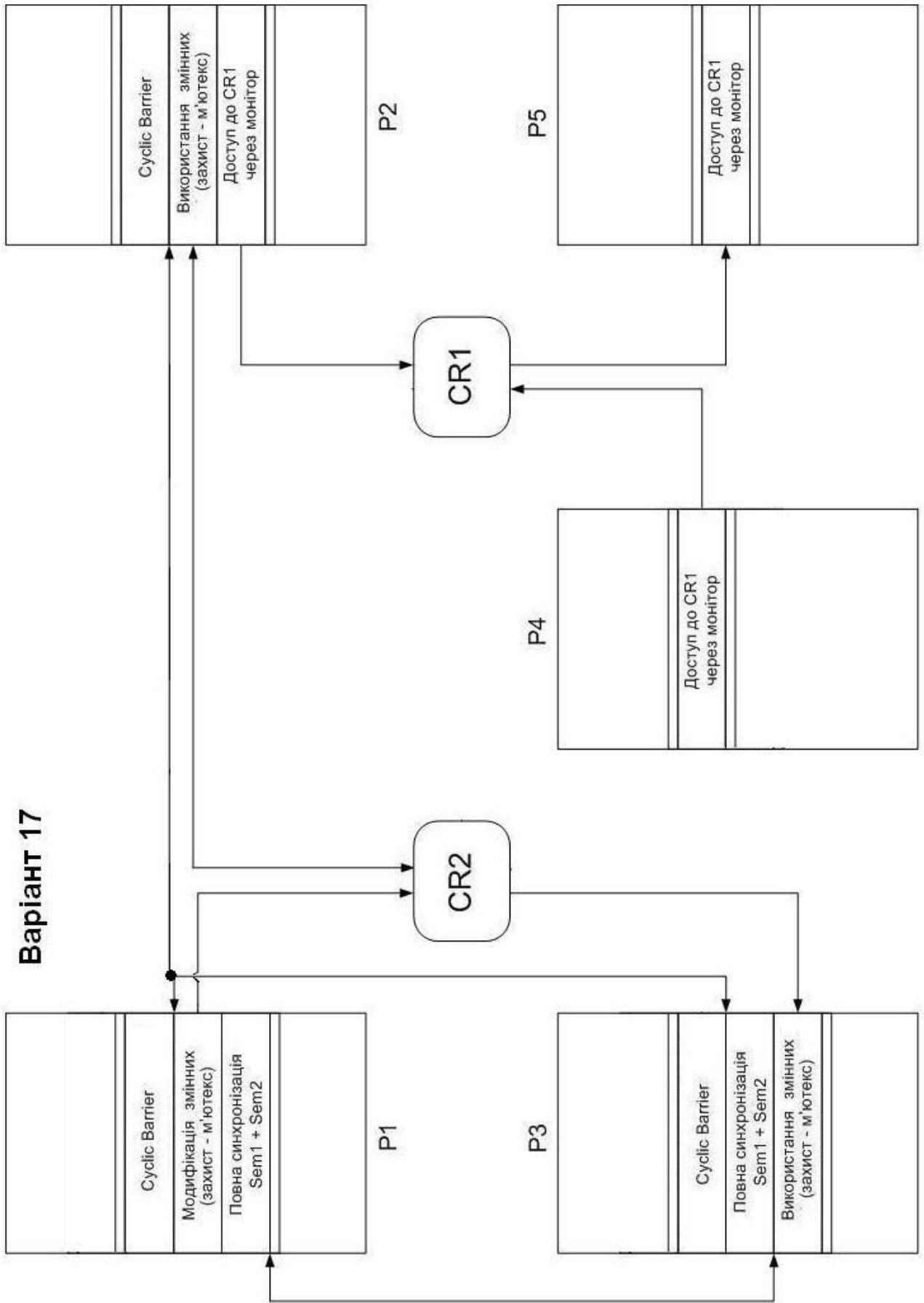


Варіант 15

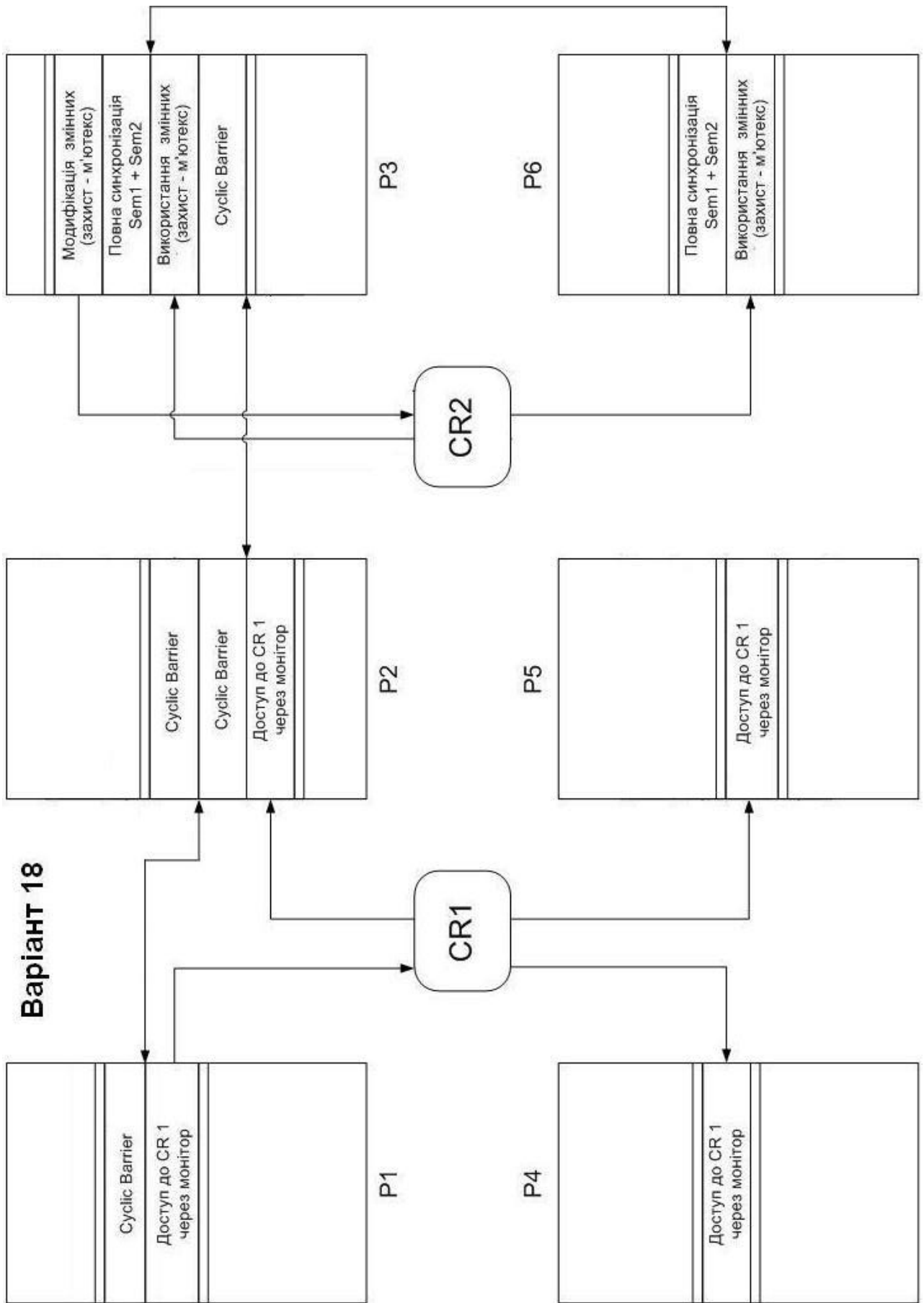


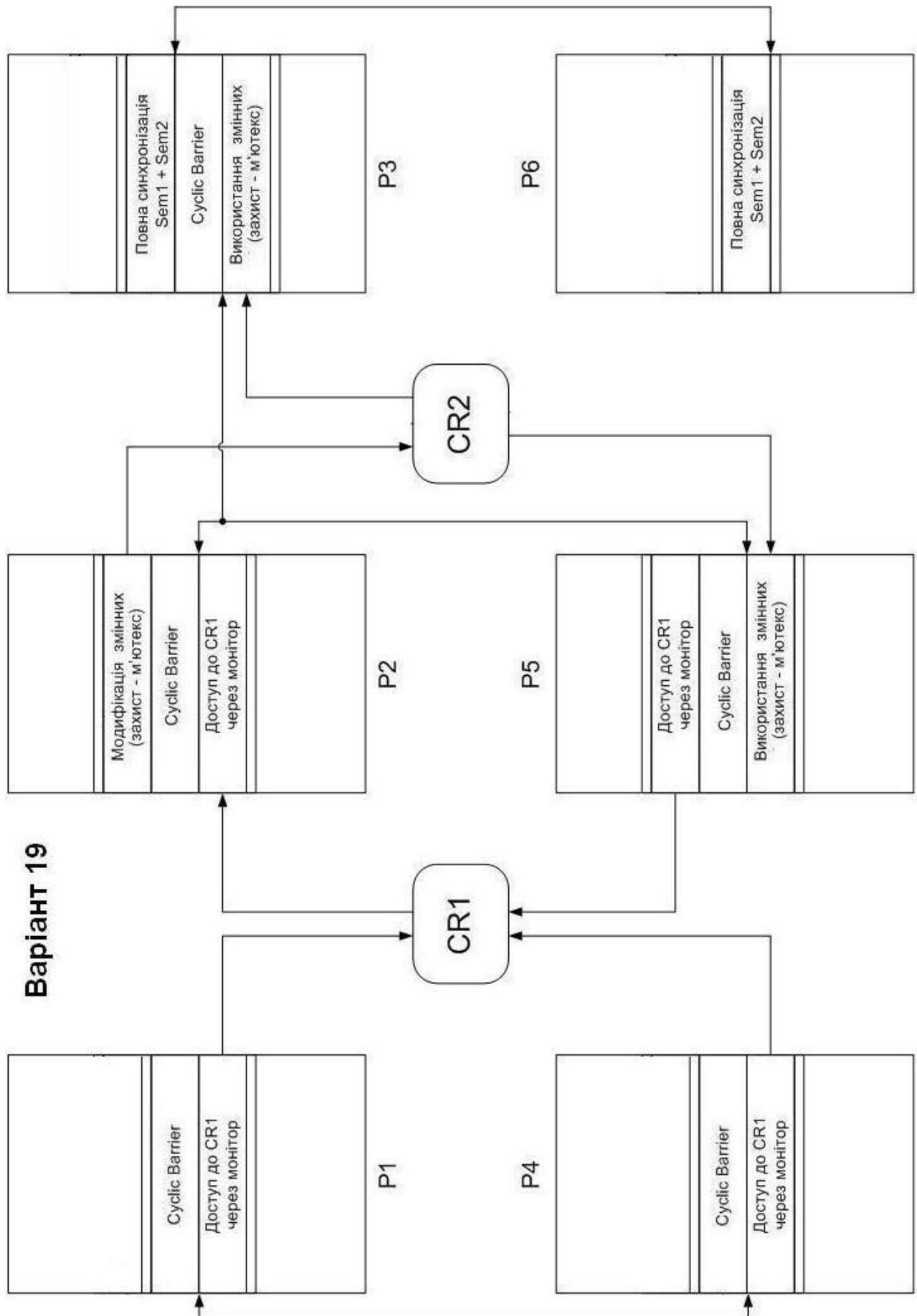


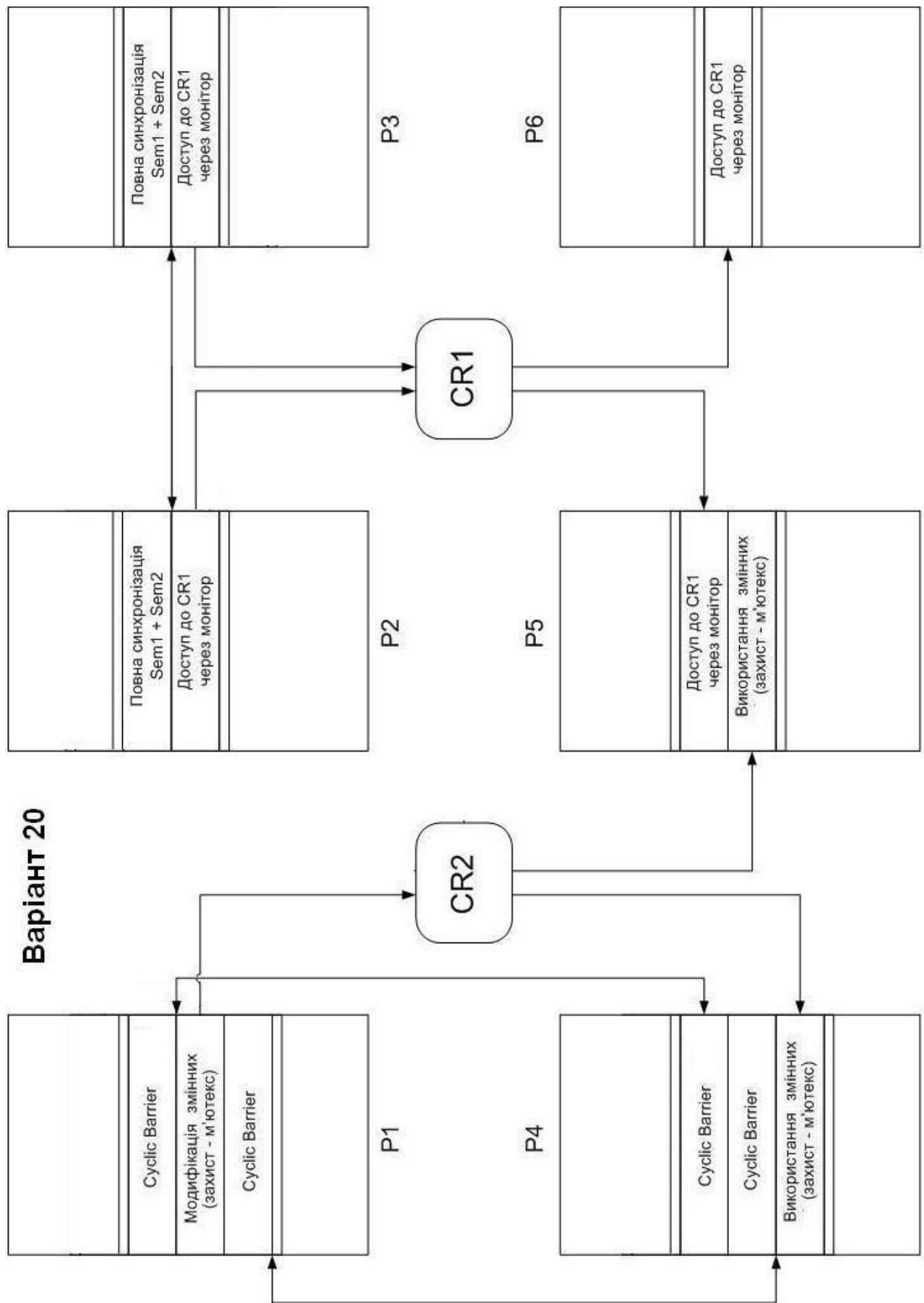
Варіант 17



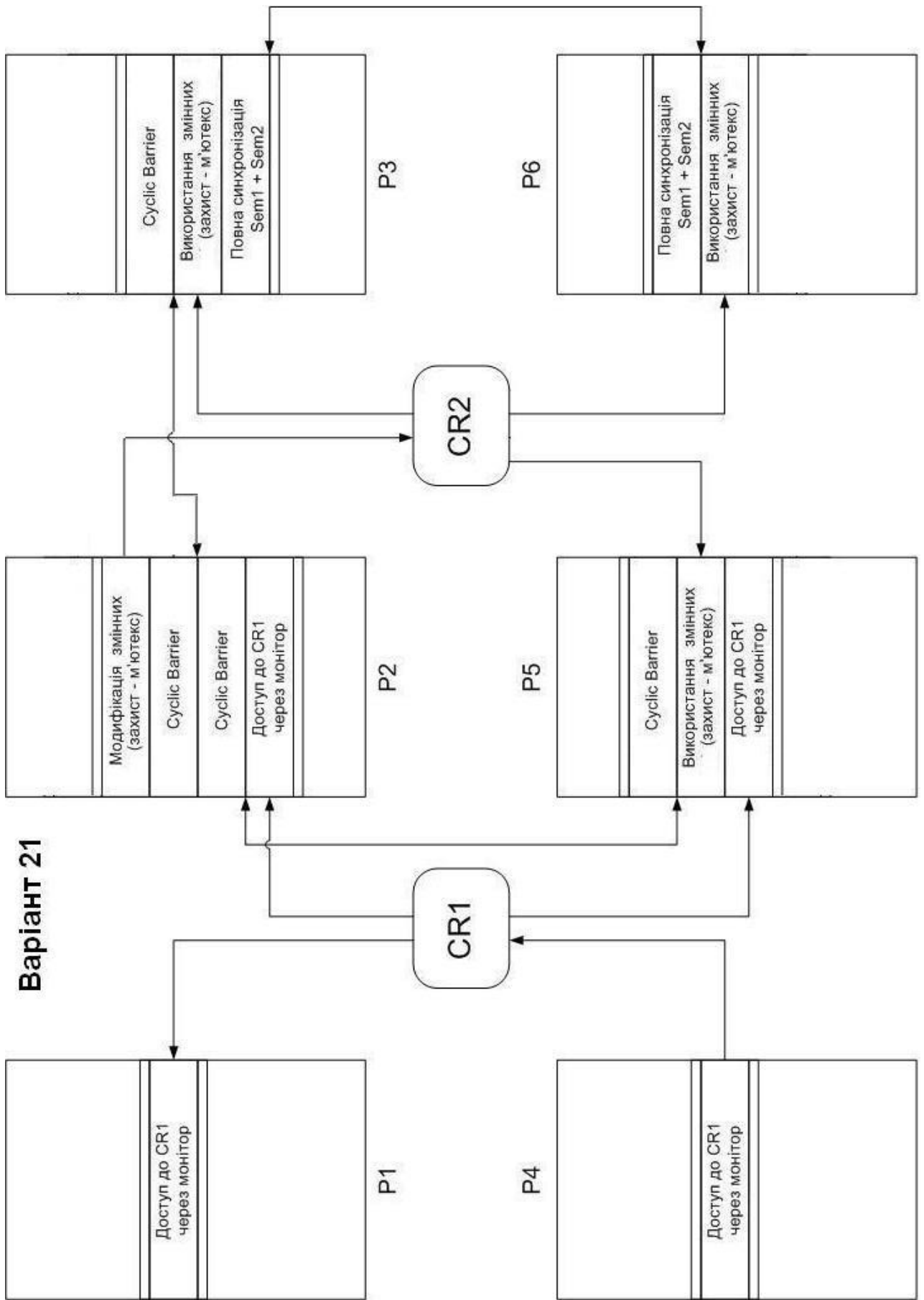
Варіант 18



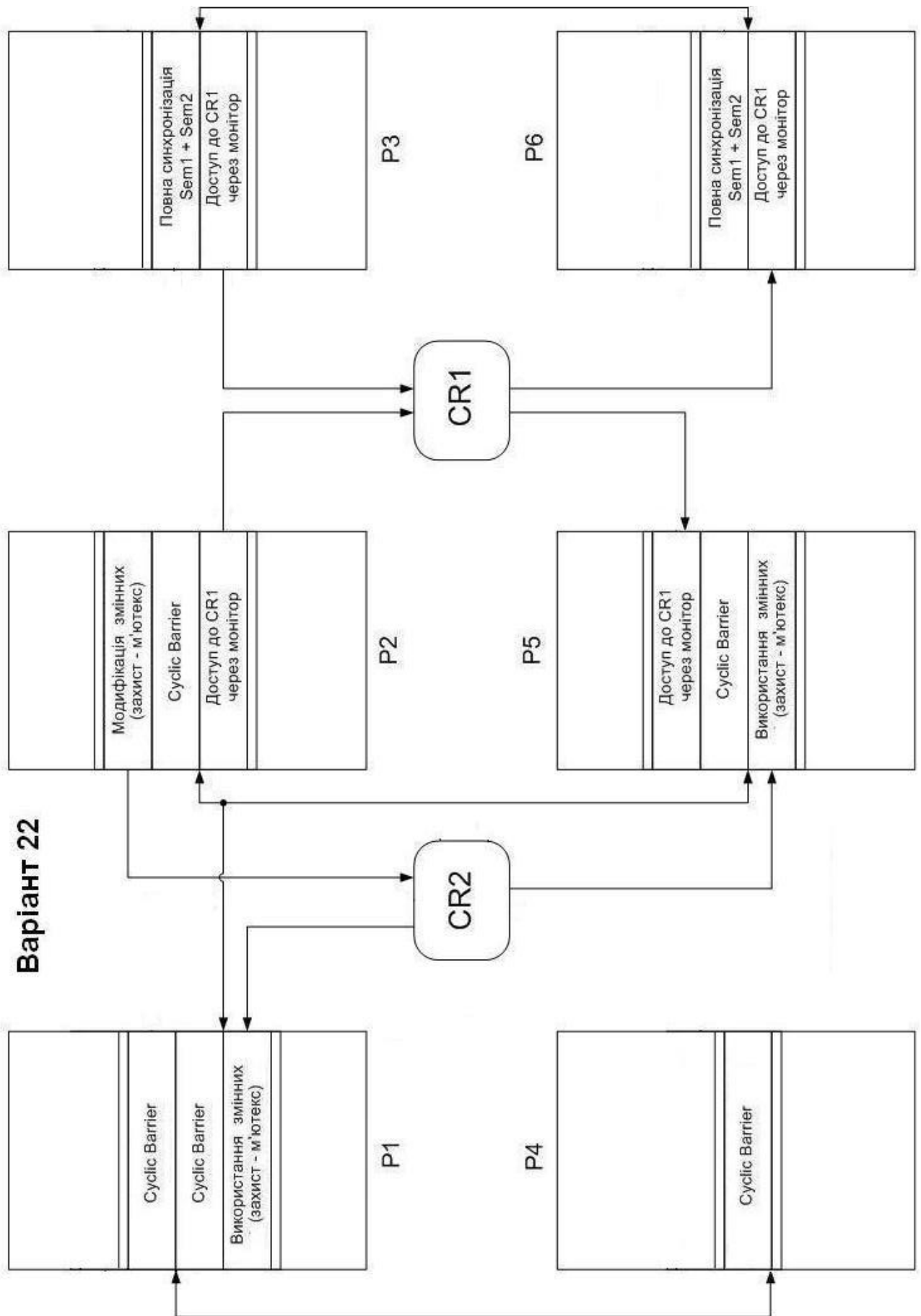




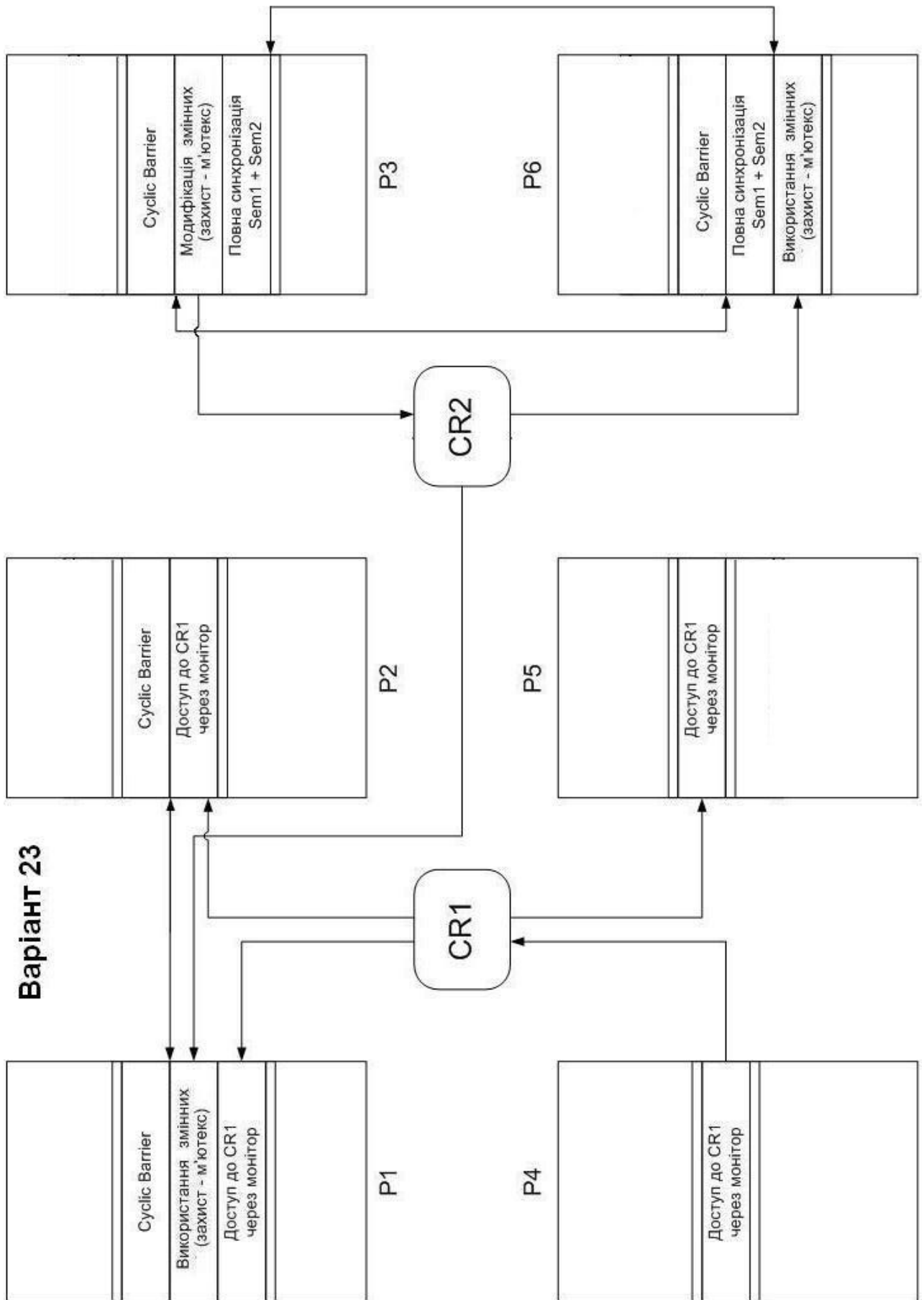
Варіант 21



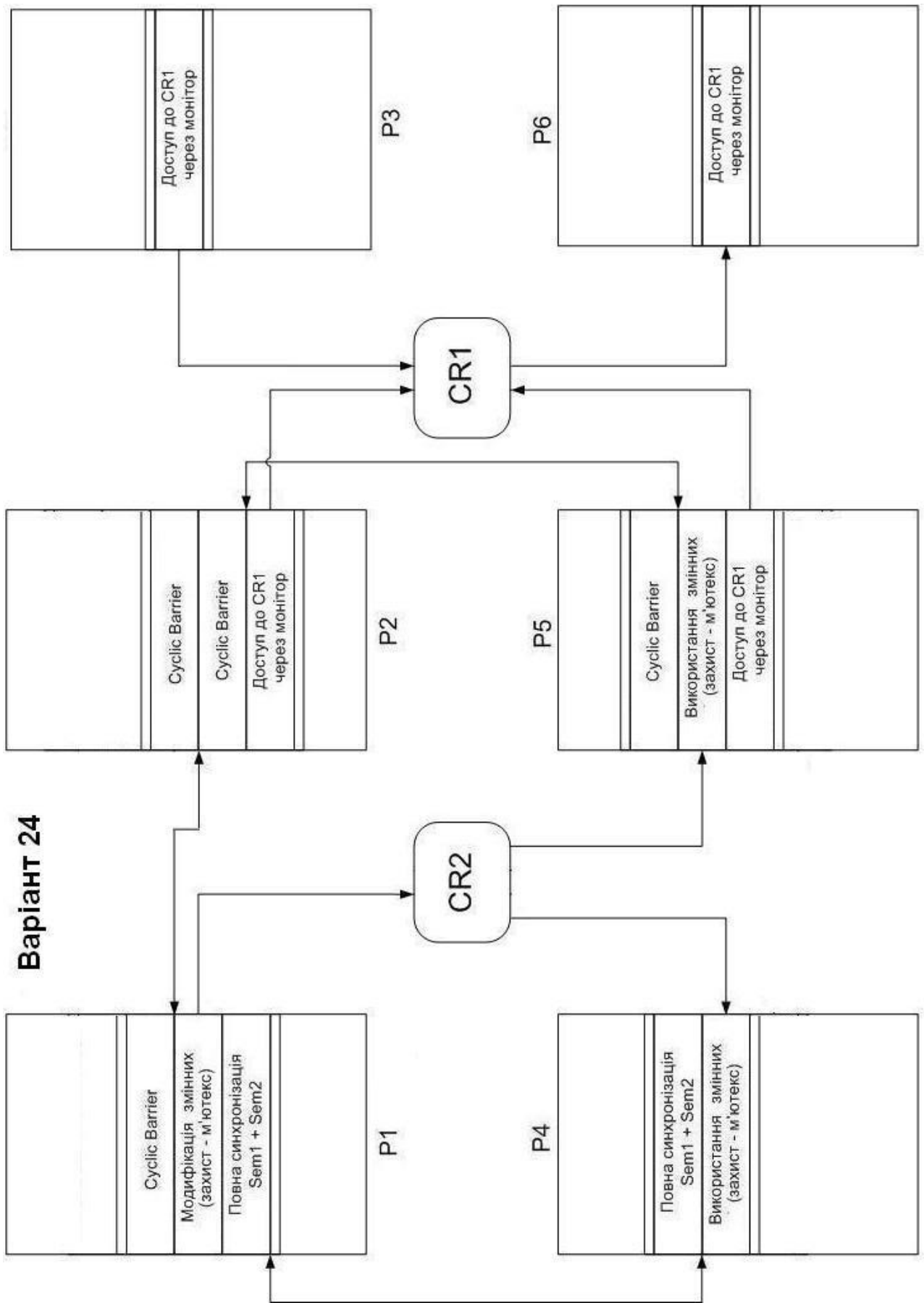
Варіант 22



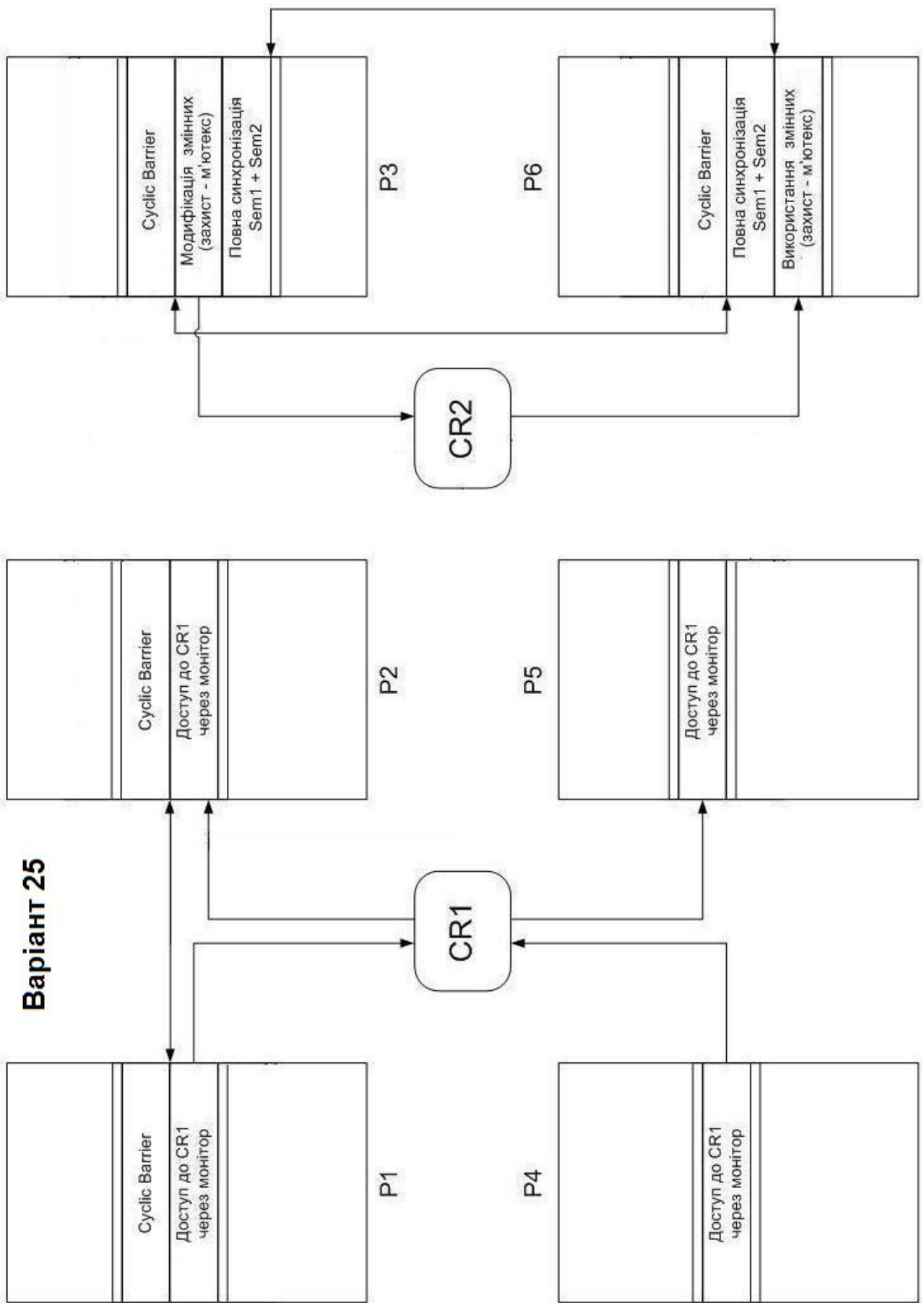
Варіант 23



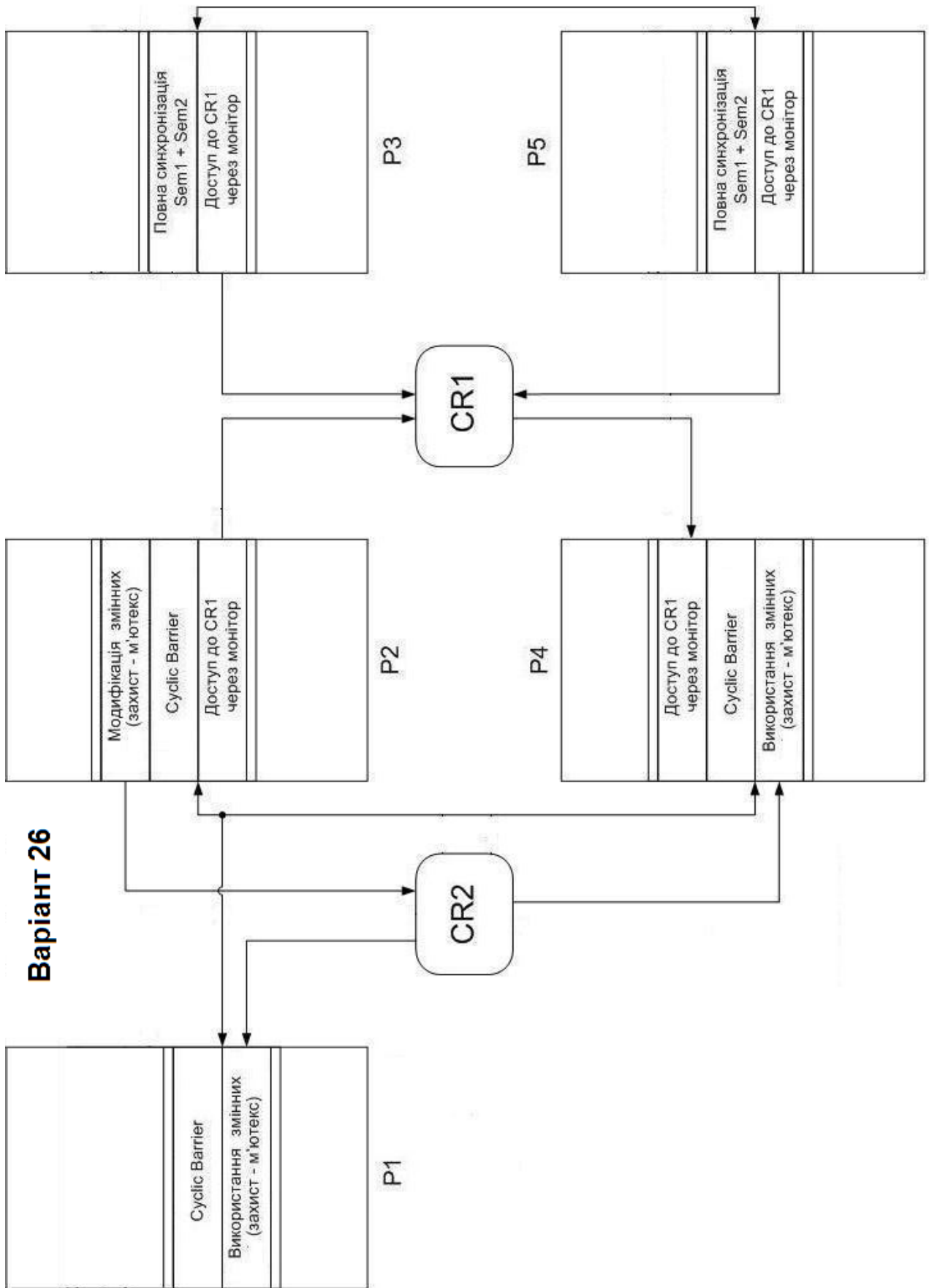
Варіант 24

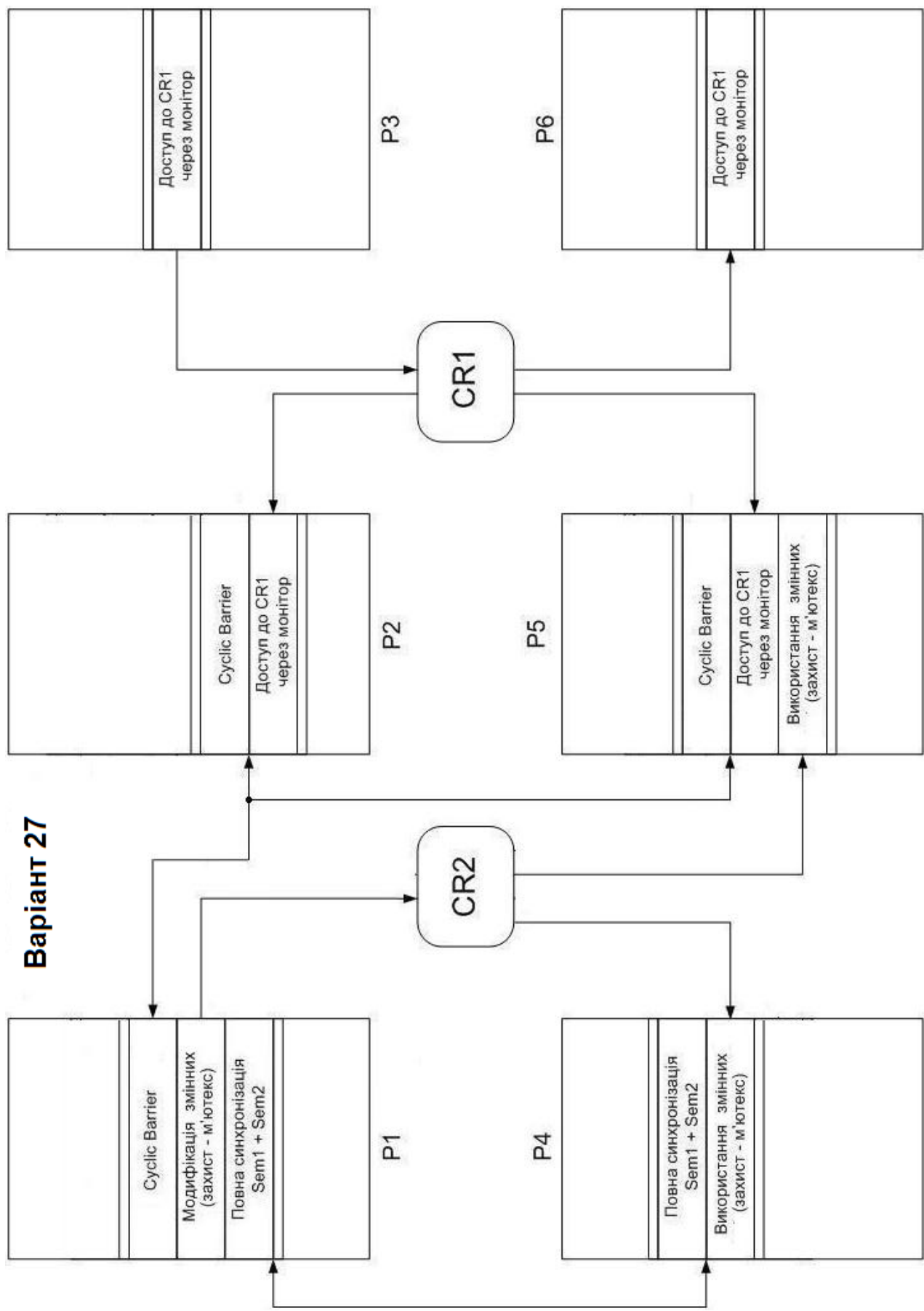


Варіант 25

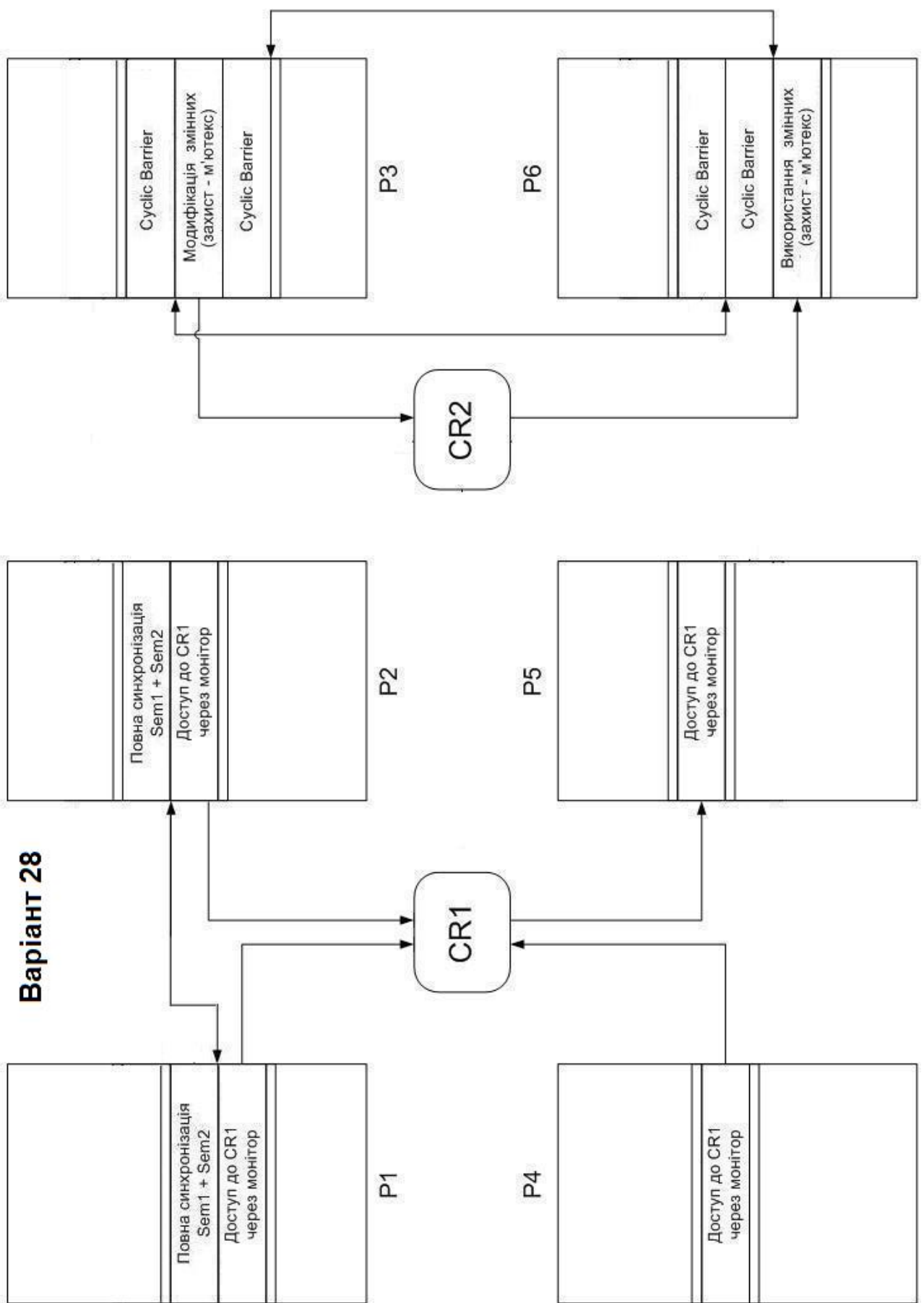


Варіант 26

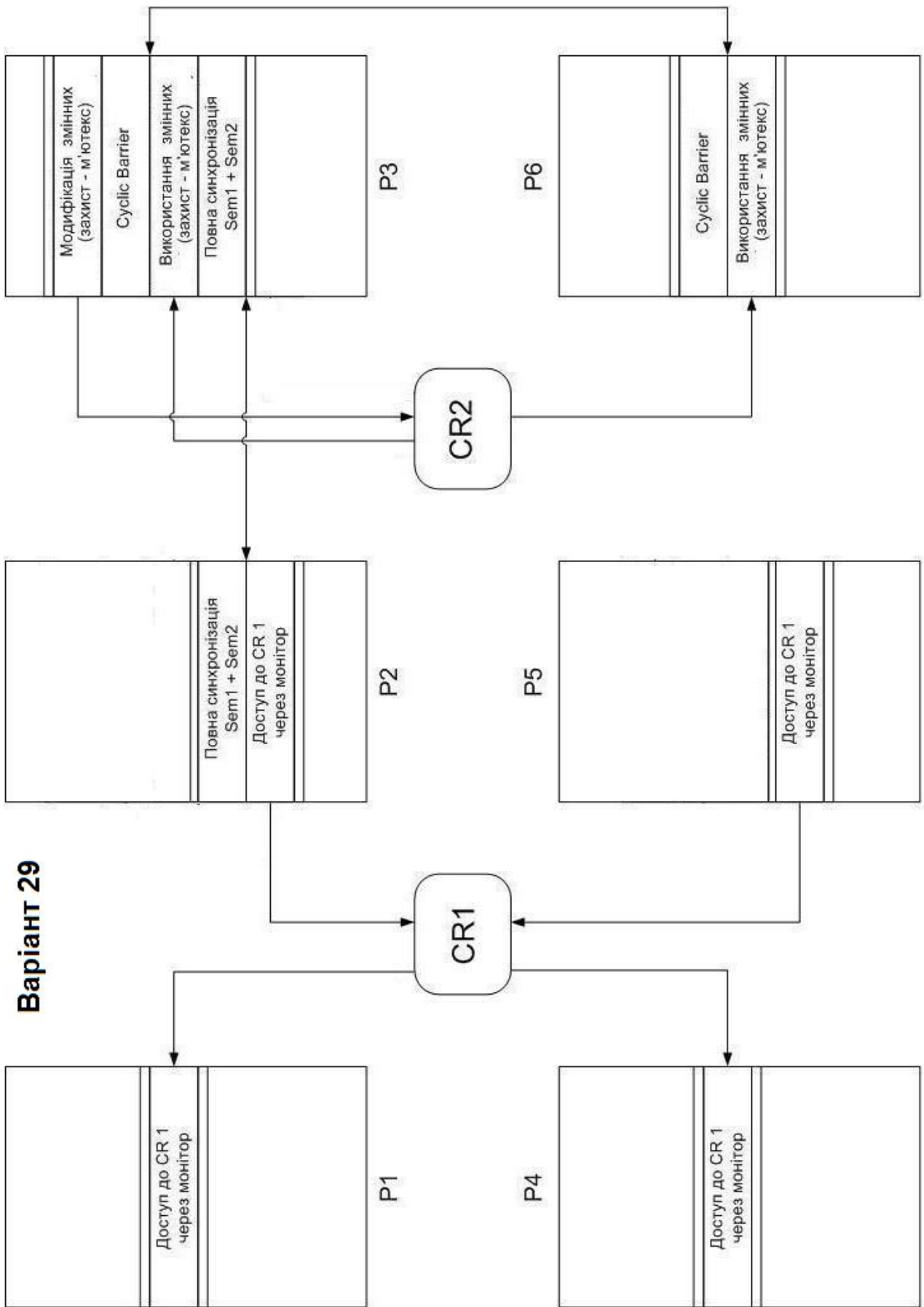




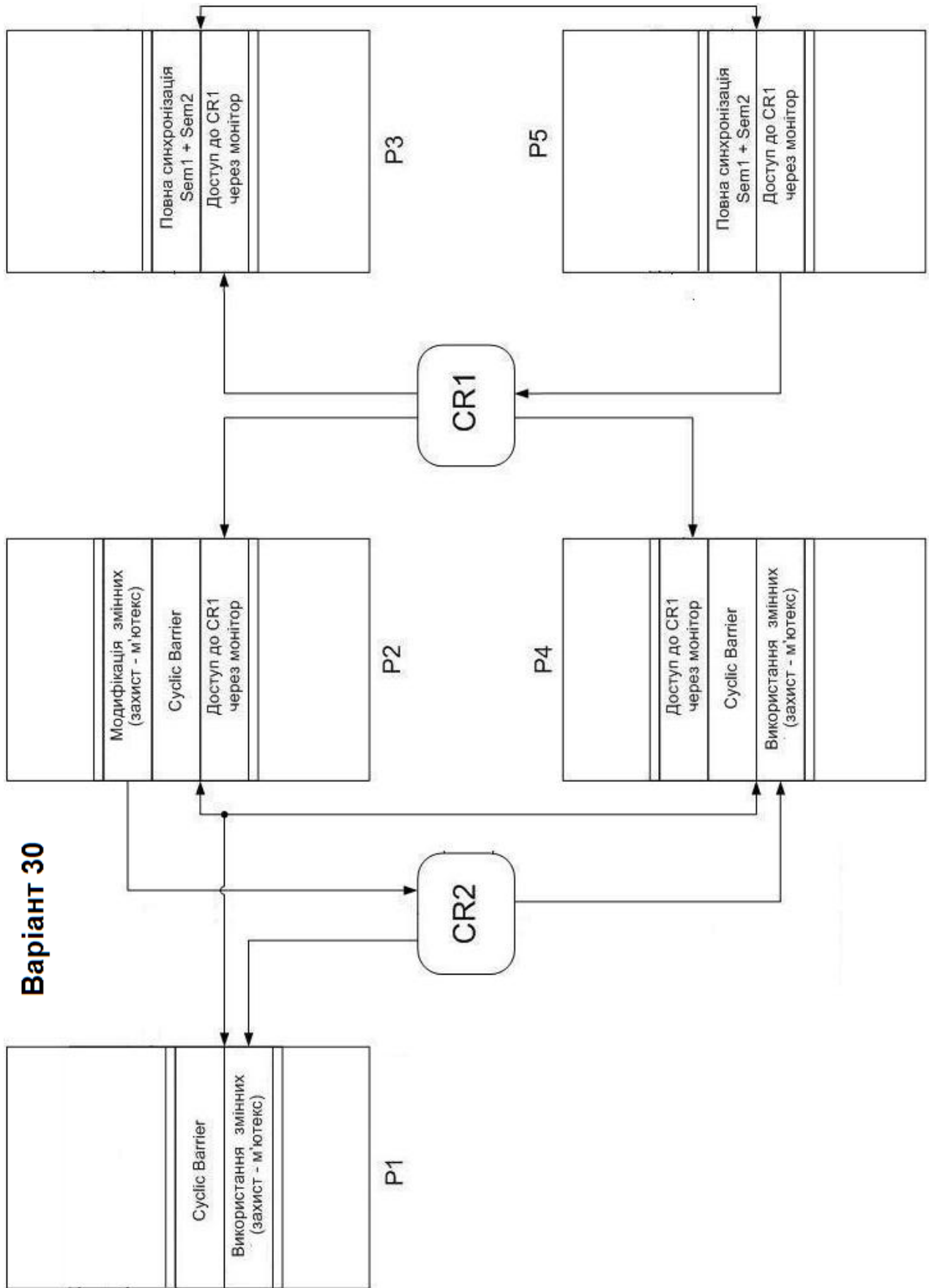
Варіант 28



Варіант 29



Варіант 30



ІНФОРМАЦІЙНІ РЕСУРСИ

1. °Youtube канал «Олександр Іванович Марченко»
<https://www.youtube.com/channel/UCerCwtA87I0yuq3tTQcK7vw>
2. Електронний кампус НТУУ «КПІ». Матеріали з дисципліни «Паралельне програмування». – Режим доступу : <http://login.kpi.ua>
3. Курс відео-лекцій на сайті кафедри СП і СКС. – Режим доступу :
<https://scs-kpi.pp.ua/course/view.php?id=125#section-1>

СПИСОК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ

1. Мельник А.О. Архітектура комп'ютера. Наукове видання. – Луцьк: Волинська обласна друкарня, 2008. – 470 с.
2. Митчел М., Оулдем Д., Самьюэл А. Программирование для Linux. Профессиональный подход.: Пер. с англ. – М.: Издательский дом «Вильямс», 2003. – 288 с.
3. Робачевский А.М. Операционная система UNIX. СПб.: БХВ – Санкт-Петербург, 1999. – 528 с., ил.
4. Sven Goldt. The Linux Programmer's Guide. Sachsendamm 47b, 10829 Berlin, Germany, 1995.
5. Герберт Шилдт. Полный справочник по Java. 7-е издание. М., СПб.: К: Издательский дом «Вильямс», 2007. – 1036 с.
6. Кен Арнольд. Язык программирования Java., СПб.: Питер, 1997. – 250 с.
7. Д.Джехани. Программирование на языке ADA., М.: Мир, 1989. – 550 с.
8. Б.Ю.Сырко, С.В.Матвеев. Программное обеспечение мульти-транспьютерных систем. М.: "Диалог-МИФИ", 1992.
9. Под ред. Г.Харпа. Транспьютеры. Архитектура и программное обеспечение. М.: "Радио и связь", 1993.
10. Кун С. Матричные процессоры на СБИС.: Пер с англ. – М.: Мир, 1991. – 672 с., ил.
11. Ричард Петерсен. LINUX: руководство по операционной системе: пер. с англ. – К.: Издательская группа BHV, 1997. – 688 с.
12. Midnight Commander. – Режим доступу :
http://linuxcommand.org/lc3_adv_mc.php

13. Р.Столмен, Р.Пеш, С.Шебс и др. Отладка с помощью GDB. – Режим доступа: <http://rus-linux.net/nlib.php?name=/MyLDP/algol/gdb/otladka-s-gdb.html>
14. Как установить Java? – Режим доступа :
http://www.java.com/ru/download/help/download_options.xml
15. Работа с Java в командной строке. – Режим доступа :
<http://habrahabr.ru/post/125210/>
16. Платформа “Eclipse”. – Режим доступа : <http://www.eclipse-3.narod.ru>
17. Быстрое учебное руководство по Java IDE NetBeans. – Режим доступа :
https://netbeans.org/kb/docs/java/quickstart_ru.html