Teoria Współbieżności Lab4 - sprawozdanie

Autor: Michał Flak

Ćwiczenie

Badanie efektywności mechanizmów synchronizacji.

Problem czytelników i pisarzy proszę rozwiązać przy pomocy:

- Semaforów
- Zmiennych warunkowych

Rozważyć przypadki z faworyzowaniem czytelników, pisarzy oraz z użyciem kolejki FIFO.

Proszę wykonać pomiary dla każdego rozwiązania dla różnej ilości czytelników (10-100) i pisarzy (od 1 do 10).

W sprawozdaniu proszę narysować wykres czasu w zależności od ilości wątków i go zinterpretować.

Implementacja

Napisano klasę abstrakcyjną, po której będą dziedziczyć poszczególne rozwiązania. Przyjmuje ona w konstruktorze ilość powtórzeń pętli czytaj/pisz, jak również długość `sleep` symulującego działanie. Klasa zawiera metodę `start` która przyjmuje w argumentach `int readers, int writers` - czyli ilość tworzonych wątków. Watki uruchamiane są w pętli jeden po drugim, po wszystkim metoda czeka na zakończenie wszystkich wątków.

```java
abstract class AbstractReadersWriters
{
    protected int repeat;
    protected int sleep;

    public AbstractReadersWriters(int repeat, int sleep) {
        this.repeat = repeat;
        this.sleep = sleep;
    }


    void start(int readers, int writers)
    {
        List<Thread> threads = new ArrayList<>();
        for (int i = 1; i <= readers; i++) {
            Thread t = new Thread(createReader());
            t.setName(String.format("#%03d", i));
            threads.add(t);
        }
        for (int i = 1; i <= writers; i++) {
            Thread t = new Thread(createWriter());
            t.setName(String.format("#%03d", i));
            threads.add(t);
        }

        for (Thread t : threads) {
            t.start();
        }
        for (Thread t : threads) {
            try {
                t.join(); // wait for all to finish
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }

    abstract Reader createReader();

    abstract Writer createWriter();

    abstract class Reader implements Runnable
    {
        @Override
        public void run()
        {
            for (int i = 0; i < repeat; i++) {
                try {
                    before();
                    read();
                    after();
                } catch (InterruptedException e) {
                    System.err.println(e.getMessage());
                }
            }
        }

        void read() throws InterruptedException
        {
            Thread.sleep(sleep);
        }

        abstract void before() throws InterruptedException;

        abstract void after() throws InterruptedException;
    }

    abstract class Writer implements Runnable
    {
        @Override
        public void run()
        {
            for (int i = 0; i < repeat; i++) {
                try {
                    before();
                    write();
                    after();
                } catch (InterruptedException e) {
                    System.err.println(e.getMessage());
                }
            }
        }

        void write() throws InterruptedException
        {
            Thread.sleep(sleep);
        }

        abstract void before() throws InterruptedException;

        abstract void after() throws InterruptedException;
    }
}
```

Main

Metoda Main tworzy pliki o nazwie `n_writers.txt`, który ma następującą zawartość:

```
nReaders RWSRP RWSWP RWSQ RWC
10       396   399   444  397
20       393   403   516  409
30       400   412   510  409
...
100      398   441   621  409
```

gdzie nReaders - liczba wątków czytelników, pozostałe kolumny przedstawiają zaś czas w milisekundach rozwiązania problemu dla odpowiednich metod.

```
public class Main {
    public static void main(String[] args) {

        int repeat = 100;
        int sleep = 1;

        Map<String, AbstractReadersWriters> solutions = new HashMap<>();
        solutions.put("RWSRP", new ReadersWritersSemReadersPreference(repeat, sleep));
        solutions.put("RWSWP", new ReadersWritersSemWritersPreference(repeat, sleep));
        solutions.put("RWSQ", new ReadersWritersSemQueue(repeat, sleep));
        solutions.put("RWC", new ReadersWritersCond(repeat, sleep));
        try {
            for(int nWriters = 1; nWriters <= 10; nWriters++){
                FileWriter myWriter = null;
                myWriter = new FileWriter(String.format("%d_writers.txt", nWriters));
                myWriter.write(String.format(
                        "%s %s %s %s %s\n",
                        "nReaders",
                        "RWSRP",
                        "RWSWP",
                        "RWSQ",
                        "RWC"
                ));

                for(int nReaders = 10; nReaders <= 100; nReaders+=10){
                    Map<String, Long> results = new HashMap<>();

                    int finalNReaders = nReaders;
                    int finalNWriters = nWriters;
                    solutions.forEach((name, solution) -> {
                        long start = System.currentTimeMillis();
                        solution.start(finalNReaders, finalNWriters);
                        long stop = System.currentTimeMillis();
                        results.put(name, stop - start);
                        System.out.printf("%s\t%d\t%d\t%d\n", name, finalNWriters, finalNReaders, stop
                    });
                    myWriter.write(String.format(
                            "%d %d %d %d %d\n",
                            nReaders,
                            results.get("RWSRP"),
                            results.get("RWSWP"),
                            results.get("RWSQ"),
                            results.get("RWC")
                    ));
                }
                myWriter.close();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

1. RWSRP - ReadersWritersSemReadersPreference

```
class ReadersWritersSemReadersPreference extends AbstractReadersWriters
{
    private int readCount = 0;
    private Semaphore resource = new Semaphore(1);
    private Semaphore rmutex = new Semaphore(1);

    public ReadersWritersSemReadersPreference(int repeat, int sleep) {
        super(repeat, sleep);
    }

    Reader createReader(){
        return new ReaderSem();
    }

    Writer createWriter(){
        return new WriterSem();
    }

    class ReaderSem extends Reader {

        @Override
        void before() throws InterruptedException {
            rmutex.acquire();
            readCount++;
            if(readCount == 1) {
                resource.acquire();
            }
            rmutex.release();
        }

        @Override
        void after() throws InterruptedException {
            rmutex.acquire();
            readCount--;
            if(readCount == 0){
                resource.release();
            }
            rmutex.release();
        }
    }

    class WriterSem extends Writer {

        @Override
        void before() throws InterruptedException {
            resource.acquire();
        }

        @Override
        void after() throws InterruptedException {
            resource.release();
        }
    }
}
```

2. RWSWP - ReadersWritersSemWritersPreference

```
class ReadersWritersSemWritersPreference extends AbstractReadersWriters
{
    private int readCount = 0;
    private int writeCount = 0;
    private Semaphore resource = new Semaphore(1);
    private Semaphore rmutex = new Semaphore(1);
    private Semaphore wmutex = new Semaphore(1);
    private Semaphore readTry = new Semaphore(1);

    public ReadersWritersSemWritersPreference(int repeat, int sleep) {
        super(repeat, sleep);
    }


    Reader createReader(){
        return new ReaderSem();
    }

    Writer createWriter(){
        return new WriterSem();
    }

    class ReaderSem extends Reader {
        @Override
        void before() throws InterruptedException {
            readTry.acquire();
            rmutex.acquire();
            readCount++;
            if(readCount == 1) {
                resource.acquire();
            }
            rmutex.release();
            readTry.release();
        }

        @Override
        void after() throws InterruptedException {
            rmutex.acquire();
            readCount--;
            if(readCount == 0){
                resource.release();
            }
            rmutex.release();
        }
    }

    class WriterSem extends Writer {

        @Override
        void before() throws InterruptedException {
            wmutex.acquire();
            writeCount++;
            if(writeCount == 1) {
                readTry.acquire();
            }
            wmutex.release();
            resource.acquire();
        }

        @Override
        void after() throws InterruptedException {
            resource.release();
            wmutex.acquire();
            writeCount--;
            if(writeCount == 0) {
                readTry.release();
            }
            wmutex.release();
        }
    }
}
```

3. RWSQ - ReadersWritersSemQueue

```
class ReadersWritersSemQueue extends AbstractReadersWriters
{
    private int readCount = 0;
    private Semaphore resourceAccess = new Semaphore(1);
    private Semaphore readCountAccess = new Semaphore(1);
    private Semaphore serviceQueue = new Semaphore(1);

    public ReadersWritersSemQueue(int repeat, int sleep) {
        super(repeat, sleep);
    }

    Reader createReader(){
        return new ReaderSem();
    }
    Writer createWriter(){
        return new WriterSem();
    }

    class ReaderSem extends Reader {

        @Override
        void before() throws InterruptedException {
            serviceQueue.acquire();           // wait in line to be serviced
            readCountAccess.acquire();        // request exclusive access to readCount
            // <ENTER>
            if (readCount == 0)         // if there are no readers already reading:
                resourceAccess.acquire();     // request resource access for readers (writers blocked)
            readCount++;                // update count of active readers
            // </ENTER>
            serviceQueue.release();           // let next in line be serviced
            readCountAccess.release();        // release access to readCount
        }

        @Override
        void after() throws InterruptedException {
            readCountAccess.acquire();        // request exclusive access to readCount
            // <EXIT>
            readCount--;                // update count of active readers
            if (readCount == 0)         // if there are no readers left:
                resourceAccess.release();     // release resource access for all
            // </EXIT>
            readCountAccess.release();        // release access to readCount
        }
    }

    class WriterSem extends Writer {

        @Override
        void before() throws InterruptedException {
            serviceQueue.acquire();           // wait in line to be serviced
            // <ENTER>
            resourceAccess.acquire();         // request exclusive access to resource
            // </ENTER>
            serviceQueue.release();           // let next in line be serviced
        }
        @Override
        void after() throws InterruptedException {
            resourceAccess.release();
        }
    }
}
```

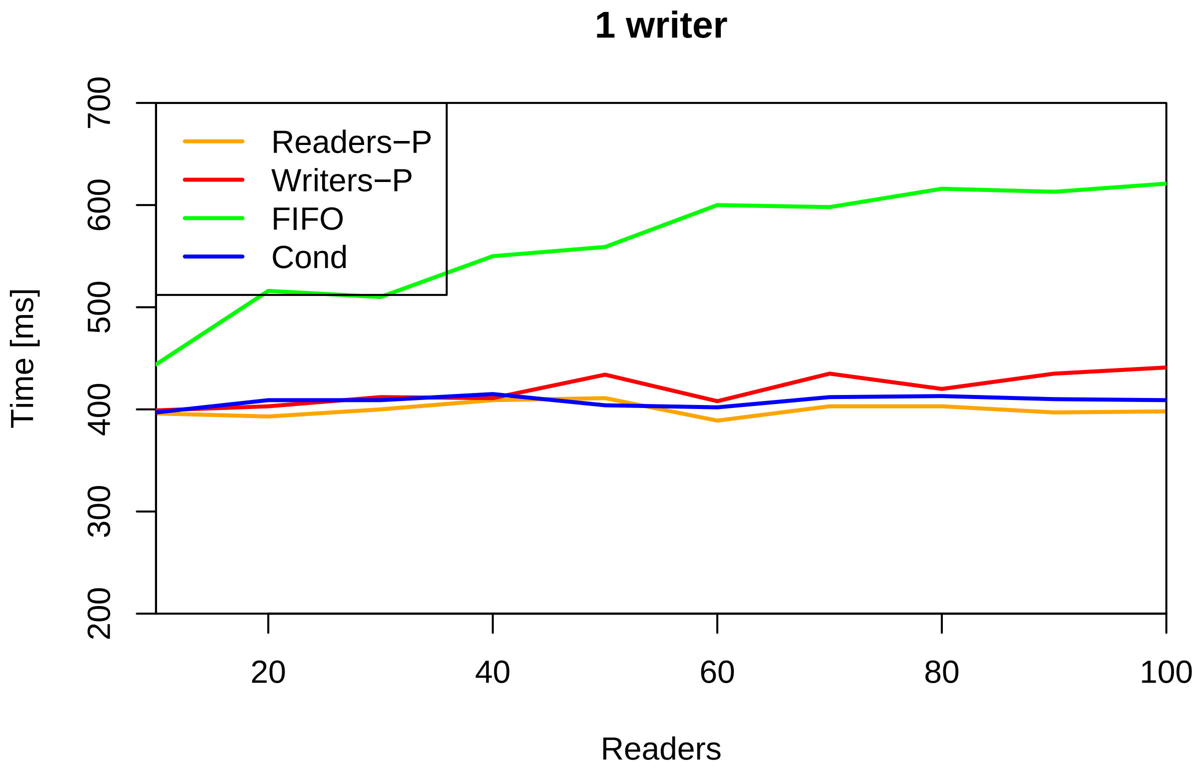
4. RWC - ReadersWritersCond

```
class ReadersWritersCond extends AbstractReadersWriters
{
    private final Lock m = new ReentrantLock();
    private final Condition turn = m.newCondition();
    private int writers = 0;
    private int writing = 0;
    private int reading = 0;

    public ReadersWritersCond(int repeat, int sleep) {
        super(repeat, sleep);
    }

    Reader createReader(){
        return new ReaderCond();
    }

    Writer createWriter(){
        return new WriterCond();
    }

    class ReaderCond extends Reader {

        @Override
        void before() throws InterruptedException {
            m.lock();
            while (0 < writers) {
                turn.await();
            }
            reading++;
            m.unlock();
        }

        @Override
        void after() throws InterruptedException {
            m.lock();
            reading--;
            turn.signalAll();
            m.unlock();
        }
    }

    class WriterCond extends Writer {

        @Override
        void before() throws InterruptedException {
            m.lock();
            writers++;
            while (0 < reading || 0 < writing) {
                turn.await();
            }
            writing++;
            m.unlock();
        }

        @Override
        void after() throws InterruptedException {
            m.lock();
            writing--;
            writers--;
            turn.signalAll();
            m.unlock();
        }
    }
}
```

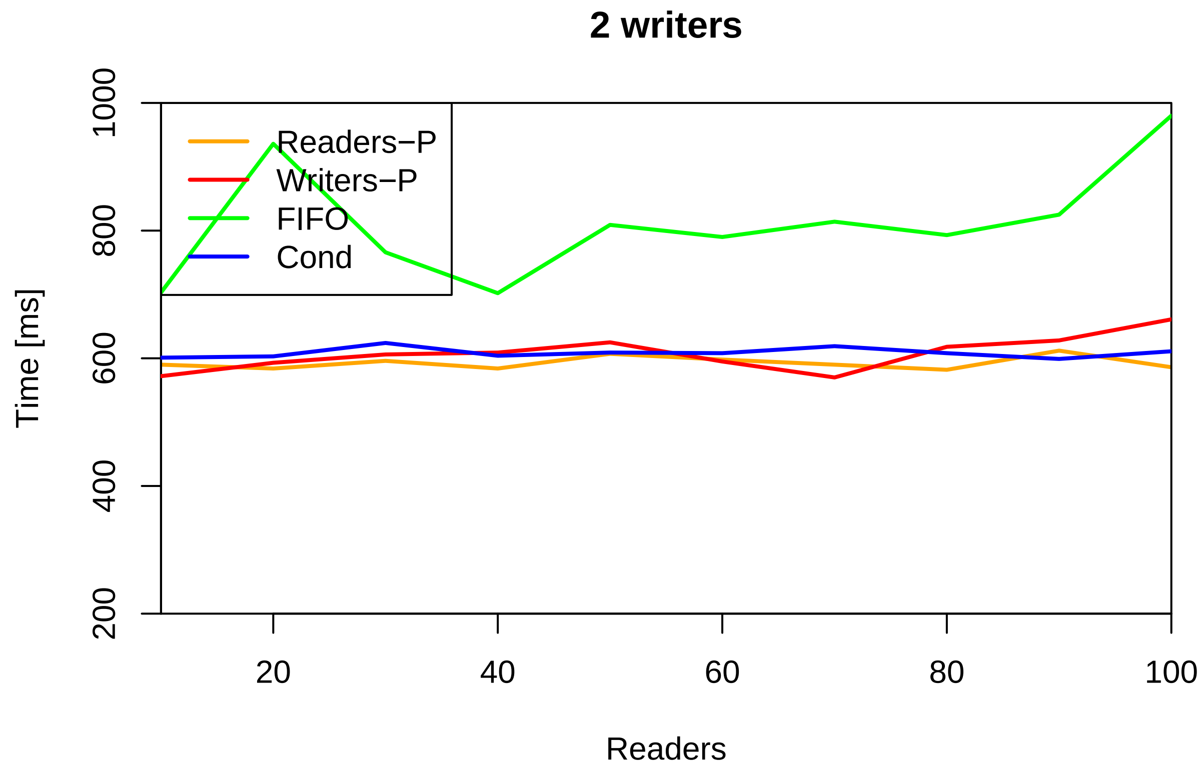
Wyniki

Wykresy czasu od ilości wątków czytelnika dla określonych ilości pisarzy:

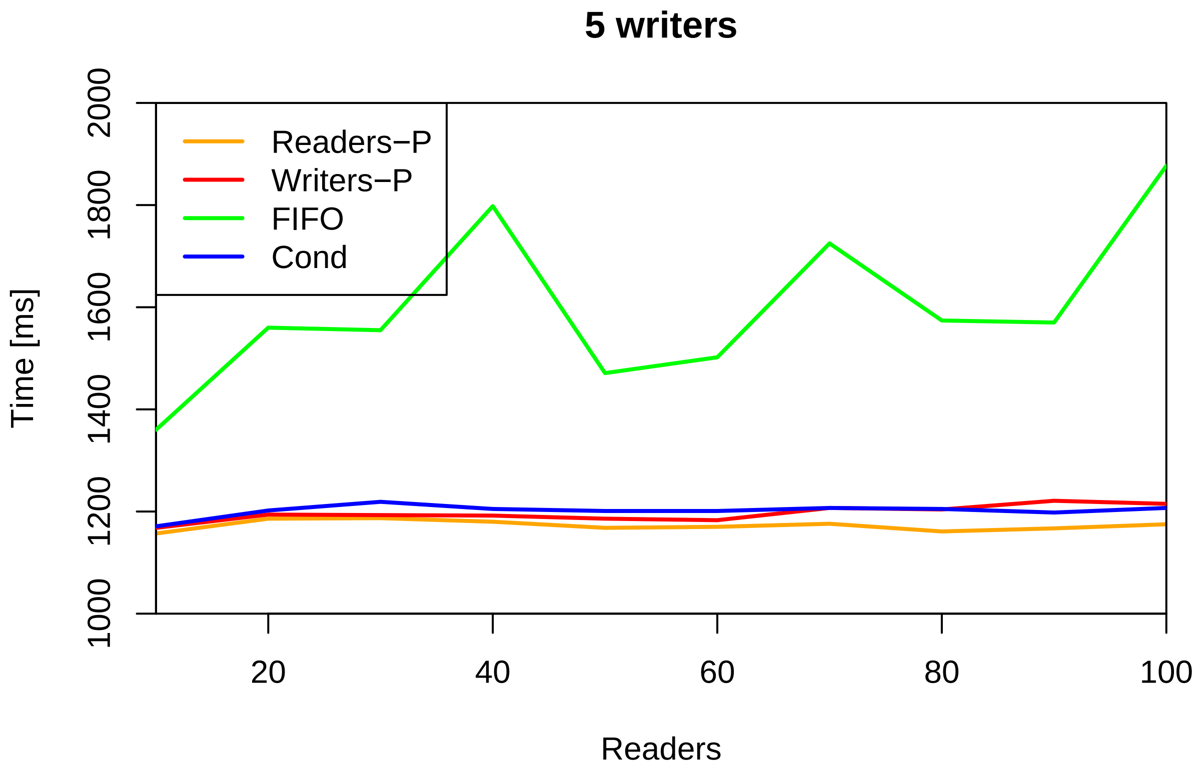
```
data <- read.table("1_writers.txt", header=T)
plot(data$nReaders, data$RWSRP, type="l", main="1 writer",
    ylim=c(200, 700), xaxs="i", yaxs="i", col="orange",
    lwd=2, xlab="Readers", ylab="Time [ms]")
lines(data$nReaders, data$RWSWP, lwd=2, col="red")
lines(data$nReaders, data$RWSQ, lwd=2, col="green")
lines(data$nReaders, data$RWC, lwd=2, col="blue")
legend("topleft", legend=c("Readers-P","Writers-P", "FIFO", "Cond"),
    lwd=c(2,2,2,2), col=c("orange","red","green","blue"))
```



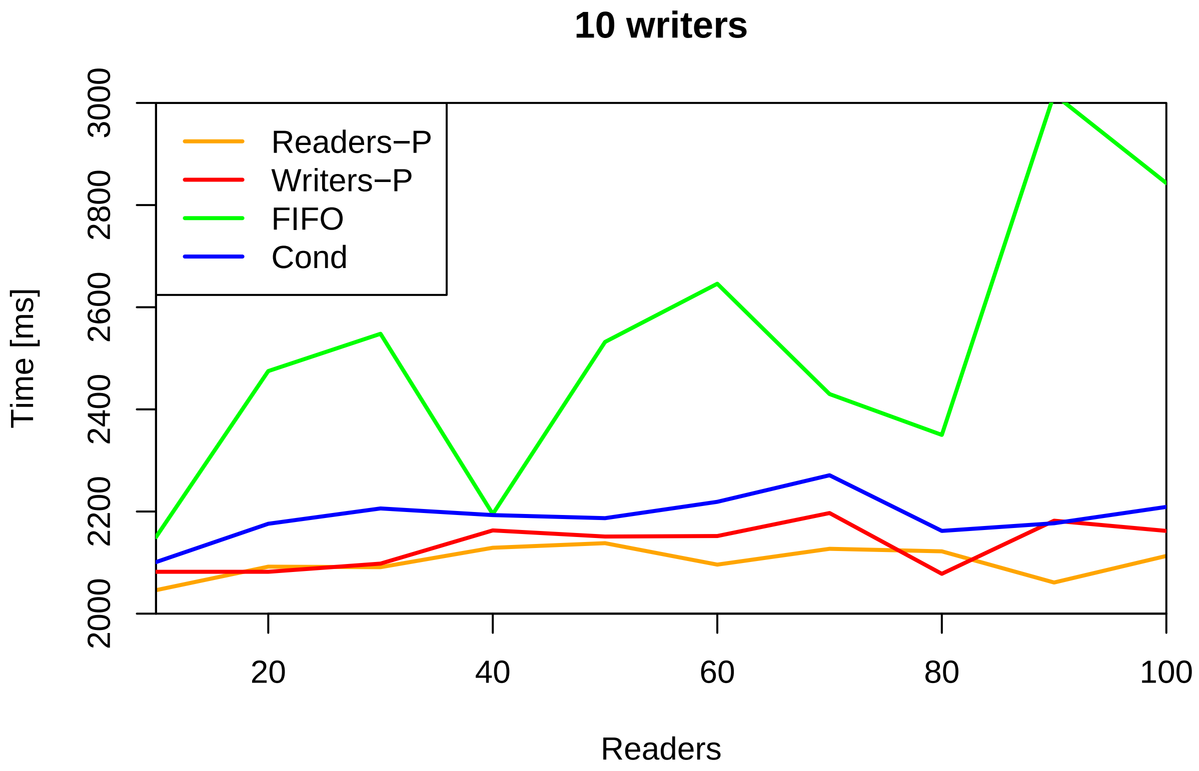
```
data <- read.table("2_writers.txt", header=T)
plot(data$nReaders, data$RWSRP, type="l", main="2 writers",
    ylim=c(200, 1000), xaxs="i", yaxs="i", col="orange",
    lwd=2, xlab="Readers", ylab="Time [ms]")
lines(data$nReaders, data$RWSWP, lwd=2, col="red")
lines(data$nReaders, data$RWSQ, lwd=2, col="green")
lines(data$nReaders, data$RWC, lwd=2, col="blue")
legend("topleft", legend=c("Readers-P","Writers-P", "FIFO", "Cond"),
    lwd=c(2,2,2,2), col=c("orange","red","green","blue"))
```



```
data <- read.table("5_writers.txt", header=T)
plot(data$nReaders, data$RWSRP, type="l", main="5 writers",
    ylim=c(1000, 2000), xaxs="i", yaxs="i", col="orange",
    lwd=2, xlab="Readers", ylab="Time [ms]")
lines(data$nReaders, data$RWSWP, lwd=2, col="red")
lines(data$nReaders, data$RWSQ, lwd=2, col="green")
lines(data$nReaders, data$RWC, lwd=2, col="blue")
legend("topleft", legend=c("Readers-P","Writers-P", "FIFO", "Cond"),
    lwd=c(2,2,2,2), col=c("orange","red","green","blue"))
```



```
data <- read.table("10_writers.txt", header=T)
plot(data$nReaders, data$RWSRP, type="l", main="10 writers",
    ylim=c(2000, 3000), xaxs="i", yaxs="i", col="orange",
    lwd=2, xlab="Readers", ylab="Time [ms]")
lines(data$nReaders, data$RWSWP, lwd=2, col="red")
lines(data$nReaders, data$RWSQ, lwd=2, col="green")
lines(data$nReaders, data$RWC, lwd=2, col="blue")
legend("topleft", legend=c("Readers-P","Writers-P", "FIFO", "Cond"),
    lwd=c(2,2,2,2), col=c("orange","red","green","blue"))
```



Wnioski

Metody:

- Semaphore readers-preference
- Semaphore writers-preference
- Conditional variables

Zachowują się w sposób bardzo zbliżony. Przy większych ilościach czytelników widzimy jednak, że najszybszy okazuje się readers-preference, a najwolniejszy writers-preference.

Zdecydowanie odbiega od nich metoda FIFO z użyciem semaforów - czasy są dużo wyższe.