Tomasz Ligęza

**Programowanie równoległe. Przetwarzanie równoległe i rozproszone.**

Sprawozdanie z laboratorium 7.

Cel zajęć:

* nabycie umiejętności pisania programóœ w języku Java z wykorzystanie puli wątków

W ramach zajęć zrealizowałem następujące kroki:

1. Utworzyłem katalog roboczy lab\_7,
2. Napisałem sekwencyjny program liczący całkę korzystając z dostarczonej klasy Calka\_callable:

Calka\_callable calka\_callable = *new* Calka\_callable(0, Math.PI, 0.001);

*double* result = calka\_callable.compute();

System.out.println("Wynik: " + result);

1. Pobrałem paczkę java\_executor\_test.tgz,
2. Na podstawie przykładu z powyższej paczki napisałem program liczący całkę na zadanym przedziale wykorzystując interfejs ExecutorService oraz klasę Executors:

*private static final int* NTHREADS = 10;

*private static final int* NUMBER\_OF\_TASKS = 50;

*private static final double* ACCURACY = 0.001;

*public static void* main(String[] args) {

ExecutorService executor = Executors.newFixedThreadPool(NTHREADS);

List<Future<Double>> results = *new* ArrayList<>(NUMBER\_OF\_TASKS);

*double* begin = 0.0;

*double* end = Math.PI;

*double* length = end - begin;

*double* step = length/NUMBER\_OF\_TASKS;

*for* (*int* i = 0; i < NUMBER\_OF\_TASKS; i++) {

Calka\_callable calka =

*new* Calka\_callable(step \* (*double*)i,

step \* (*double*)(i+1),

ACCURACY);

Future<Double> future = executor.submit(calka);

results.add(future);

}

double result = 0.0;

*for*(*var* future : results) {

*try* {

result += future.get();

} *catch* (ExecutionException | InterruptedException e) {

e.printStackTrace();

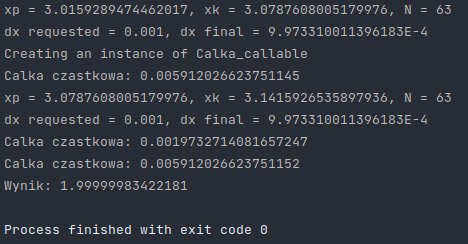
}

}

executor.shutdown();

System.out.println("Wynik: " + result);

Zrzut ekranu z fragmentem wypisu zawierającego wynik:



1. Stworzyłem kolejny katalog roboczy - lab\_7\_fork,
2. Uzupełniłem szkielet programu do sortowania oraz wykorzystując klasę ForkJoinPool uruchomiłem wielowątkowy program sortujący metodą merge-sort:

Metoda compute() z klasy DivideTask:

*protected int*[] compute() {

*if*(arrayToDivide.length == 1)

*return* arrayToDivide;

DivideTask task1 = *new* DivideTask(Arrays.copyOfRange(arrayToDivide,

0,

arrayToDivide.length/2));

DivideTask task2 = *new* DivideTask(Arrays.copyOfRange(arrayToDivide,

arrayToDivide.length/2,

arrayToDivide.length));

task1.fork();

task2.fork();

*int*[] tab1 = task1.join();

*int*[] tab2 = task2.join();

*int*[] scal\_tab = *new int*[arrayToDivide.length];

scal\_tab(tab1, tab2, scal\_tab);

*return* scal\_tab;

}

Metoda main():

*int[] numbers = {2, 4, 1, 2, -2, 9, -10};*

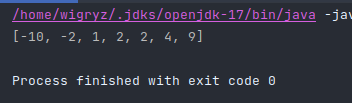
*DivideTask task = new DivideTask(numbers);*

*ForkJoinPool forkJoinPool = new ForkJoinPool();*

*forkJoinPool.execute(task);*

*System.out.println(Arrays.toString(task.join()));*

Wywołanie programu:



1. Zmodyfikowałem program do obliczania histogramu tak, żeby korzystał z puli wątków:

*int* num\_threads = Runtime.getRuntime().availableProcessors();

ExecutorService executor = Executors.newFixedThreadPool(num\_threads);

*int* charsForThread = (*int*) Math.ceil((*double*)NUMBER\_OF\_CHARS/num\_threads);

*for* (*int* i = 0; i < num\_threads; i++) {

*int* start = i \* charsForThread;

*int* end = Math.min(((i + 1) \* charsForThread), NUMBER\_OF\_CHARS);

executor.submit(*new* WatekDec(start, end, obraz\_1));

}

executor.shutdown();

*while*(!executor.isTerminated()) {}

System.out.println("Czy oba histogramy sie zgadzaja? " +

obraz\_1.checkBothHistogramsDec());

Zrzut ekranu z wykonania programu:



Wnioski:

Pule wątków bardzo ułatwiają pracę z wątkami. Dobrze prezentuje to ćwiczenie z sortowaniem przez scalanie. Rekursywne uruchamianie kolejnych wątków liczących podzadania byłoby dużo bardziej skomplikowane niż to z wykorzystaniem kasy ForkJoinPool.

Pule wątków umożliwiają nam pominięcie mozolnej pracy związanej z tworzeniem wątków, oczekiwaniem na zakończenie ich pracy oraz zarządzanie nimi. Dzięki temu możemy oddzielić tworzenie zadań od zarządzania wątkami.