Sprawozdanie nr 3

Data ćwiczeń: 09.04.2015

Data oddania sprawozdania: 23.04.2015

* Tematyka

Na trzecich zajęciach laboratoryjnych poruszana była problematyka związana ze wspólnym buforem dla n wątków, które do bufora zapisywały dane i dla m wątków, które te dane odczytywały.

Problem definiuje się w dość oczywisty sposób: z bufora jednocześnie można albo czytać, albo zapisywać do niego dane. Należy również wykluczyć sytuację, w której wątek odpowiedzialny za czytanie danych odczytuje dane z pustego bufora. O ile język Java jest na takie sytuacje odporny, to inne języki, np C wysłałyby signał SIGSEGV powodujący zakończenie programu komunikatem z błędem.

* Zadanie

Na zajęciach należało zaimplementować dwa zadania. Jako pierwsze był to problem ograniczonego bufora (*producenci* nie zapisują więcej danych niż rozmiar bufora, a *konsumenci* nie odczytują z niego więcej niż się w nim znajduje). Drugie zadanie polegało na implementacji podobnej problematyki, ale tym razem z wykorzystaniem bufora.

* Kod zadania 1

*/\*\*  
 \* Created by Patryk on 2015-04-09.  
 \*/***public class** Lab3Zad1 {  
 **public static void** main(String[] args) **throws** InterruptedException {  
 Buffer buf = **new** Buffer(100);  
 **int** n1 = 5;  
 **int** n2 = 15;  
  
 **int** prodIterations = 300;  
 **int** consIterations = 100;  
  
 **if**(n1\*prodIterations != n2\*consIterations) {  
 **throw new** WrongParametersException(**"Number of goods produced "** +  
 **"is not equal with number of goods consumed. Check**

**the params!"**);  
 }  
  
 ExecutorService service = Executors.*newFixedThreadPool*(n1 + n2);  
  
 **for**(**int** i=1; i<=n1; i++) {  
 service.submit(**new** Producer(buf, prodIterations));  
 }  
  
 **for**(**int** i=1; i<=n2; i++) {  
 service.submit(**new** Consumer(buf, consIterations));  
 }  
  
 service.shutdown();  
 }  
}

*/\*\*  
 \* Created by Patryk on 2015-04-09.  
 \*/***public class** Producer **extends** Thread {  
 **private** Buffer **\_buf**;  
 **private int iterations**;  
  
 **public** Producer(Buffer buf, **int** iterations) {  
 **this**.**\_buf** = buf;  
 **this**.**iterations** = iterations;  
 }  
  
 **public void** run() {  
 **for** (**int** i = 0; i < **iterations**; ++i) {  
 **\_buf**.put(i);  
 **try** {  
 *sleep*((**int**) (Math.*random*() \* 100));  
 } **catch** (InterruptedException e) {  
 System.***out***.println(e.getMessage());  
 }  
 }  
 }  
}

*/\*\*  
 \* Created by Patryk on 2015-04-09.  
 \*/***public class** Consumer **extends** Thread {  
 **private** Buffer **\_buf**;  
 **private int iterations**;  
  
 **public** Consumer(Buffer buf, **int** iterations) {  
 **this**.**\_buf** = buf;  
 **this**.**iterations** = iterations;  
 }  
  
 **public void** run() {  
 **for** (**int** i = 0; i < **iterations**; ++i) {  
 **\_buf**.get();  
 **try** {  
 *sleep*((**int**) (Math.*random*() \* 100));  
 } **catch** (InterruptedException e) {  
 System.***out***.println(e.getMessage());  
 }  
 }  
 }  
}

*/\*\*  
 \* Created by Patryk on 2015-04-09.  
 \*/***public class** Buffer {  
 **private** List<Integer> **buf** = **new** ArrayList<Integer>();  
 **private int M**;  
  
 **public** Buffer(**int** m) {  
 **this**.**M** = m;  
 }  
  
 **public synchronized void** put(**int** i) {  
 **while**(**buf**.size() >= **M**) {  
 **try** {  
 System.***out***.println(**"\tProducer waits."**);  
 wait();  
 } **catch**(InterruptedException iexp) {  
 System.***out***.println(iexp.getMessage());  
 }  
 }  
  
 System.***out***.println(**"Producer puts "** + i);  
 **buf**.add(i);  
 notify();  
 }

**public synchronized int** get() {  
 **while**(**buf**.size() == 0) {  
 **try** {  
 System.***out***.println(**"\tConsumer waits."**);  
 wait();  
 } **catch**(InterruptedException iexp) {  
 System.***out***.println(iexp.getMessage());  
 }  
 }  
  
 **int** index = **new** Random().nextInt(**buf**.size());  
 **int** returnVal = **buf**.get(index);  
 **buf**.remove(index);  
 notify();  
 System.***out***.println(**"Consumer gets "** + returnVal);  
 **return** returnVal;  
 }  
}

* Kod zadania 2

*/\*\*  
 \* Created by Patryk on 2015-04-09.  
 \*/***public class** Lab3Zad2 {  
 **public static void** main(String[] args) **throws** InterruptedException {  
 Semaphore semaphore = **new** Semaphore();  
 Buffer2 buf = **new** Buffer2(100);  
  
 **int** n1 = 1;  
 **int** n2 = 1;  
 **int** prodIterations = 100;  
 **int** consIterations = 100;  
  
 **if**(n1\*prodIterations != n2\*consIterations) {  
 **throw new** WrongParametersException2(**"Number of goods produced is not equal with number of goods consumed. Check the params!"**);  
 }  
  
 ExecutorService service = Executors.*newFixedThreadPool*(n1 + n2);  
  
 **for**(**int** i=1; i<=n1; i++) {  
 service.submit(**new** Producer2(buf, i, prodIterations, semaphore));  
 }  
  
 **for**(**int** i=1; i<=n2; i++) {  
 service.submit(**new** Consumer2(buf, i, consIterations, semaphore));  
 }  
  
 service.shutdown();  
 }  
}

*/\*\*  
 \* Created by Patryk on 2015-04-09.  
 \*/***public class** Producer2 **extends** Thread {  
 **private** Buffer2 **\_buf**;  
 **private int num**;  
 **private int iterations**;  
 **private** Semaphore **semaphore**;  
  
 **public** Producer2(Buffer2 buf, **int** num, **int** iterations, Semaphore semaphore) {  
 **this**.**\_buf** = buf;  
 **this**.**num** = num;  
 **this**.**iterations** = iterations;  
 **this**.**semaphore** = semaphore;  
 }  
  
 **public void** run() {  
 **for** (**int** i = 0; i < **iterations**; ++i) {  
 **try** {  
 **this**.**semaphore**.P();  
 **\_buf**.put(i);  
 *sleep*((**int**) (Math.*random*() \* 100));  
 } **catch** (InterruptedException e) {  
 System.***out***.println(e.getMessage());  
 } **finally** {  
 **this**.**semaphore**.V();  
 }  
 }  
 }  
}

*/\*\*  
 \* Created by Patryk on 2015-04-09.  
 \*/***public class** Consumer2 **extends** Thread {  
 **private** Buffer2 **\_buf**;  
 **private int num**;  
 **private int iterations**;  
 **private** Semaphore **semaphore**;  
  
 **public** Consumer2(Buffer2 buf, **int** num, **int** iterations, Semaphore semaphore) {  
 **this**.**\_buf** = buf;  
 **this**.**num** = num;  
 **this**.**iterations** = iterations;  
 **this**.**semaphore** = semaphore;  
 }  
  
 **public void** run() {  
 **for** (**int** i = 0; i < **iterations**; ++i) {  
 **try** {  
 **this**.**semaphore**.P();  
 **\_buf**.get();  
 *sleep*((**int**) (Math.*random*() \* 100));  
 } **catch** (InterruptedException e) {  
 System.***out***.println(e.getMessage());  
 } **finally** {  
 **this**.**semaphore**.V();  
 }  
 }  
 }  
}

*/\*\*  
 \* Created by Patryk on 2015-04-09.  
 \*/***public class** Buffer2 {  
 **private** List<Integer> **buf** = **new** ArrayList<Integer>();  
 **private int M**;  
  
 **public** Buffer2(**int** m) {  
 **this**.**M** = m;  
 }  
  
 **public synchronized void** put(**int** i) {  
 **while**(**buf**.size() >= **M**) {  
 **try** {  
 System.***out***.println(**"\tProducer waits."**);  
 wait();  
 } **catch**(InterruptedException iexp) {  
 System.***out***.println(iexp.getMessage());  
 }  
 }  
  
 System.***out***.println(**"Producer puts "** + i);  
 **buf**.add(i);  
 notify();  
 }  
  
 **public synchronized int** get() {  
 **while**(**buf**.size() == 0) {  
 **try** {  
 System.***out***.println(**"\tConsumer waits."**);  
 wait();  
 } **catch**(InterruptedException iexp) {  
 System.***out***.println(iexp.getMessage());  
 }  
 }  
  
 **int** index = **new** Random().nextInt(**buf**.size());  
 **int** returnVal = **buf**.get(index);  
 **buf**.remove(index);  
 notify();  
 System.***out***.println(**"Consumer gets "** + returnVal);  
 **return** returnVal;  
 }  
}

*/\*\*  
 \* Created by Patryk on 2015-04-09.  
 \*/***public class** Semaphore {  
 **private boolean \_stan**;  
 **private int \_czeka**;  
  
 **public** Semaphore() {  
 **this**.**\_stan** = **true**;  
 **this**.**\_czeka** = 0;  
 }  
  
 **public synchronized void** P() **throws** InterruptedException {  
 *//opuszczenie semafora - acquire* **\_czeka**++;  
 **while**(!**\_stan**) {  
 **try** {  
 wait();  
 } **catch** (InterruptedException ie) {  
 System.***out***.println(ie.getMessage());  
 }  
 }  
 **\_czeka**--;  
 **\_stan** = **false**;  
 }  
  
 **public synchronized void** V() {  
 *//podniesienie semafora - release* **if**(**\_czeka** > 0) {  
 **this**.notify();  
 }  
 **\_stan** = **true**;  
 }  
}