Національний Технічний Університет України

«Київський Політехнічний Інститут імені Ігоря Сікорського» Інститут Прикладного Системного Аналізу

Кафедра Системного Проектування

Паралельні обчислення

Лабораторна робота №2

**Роботу виконав:** Нікітін Б. Д.

Група: ДА­81

**Перевірив:**

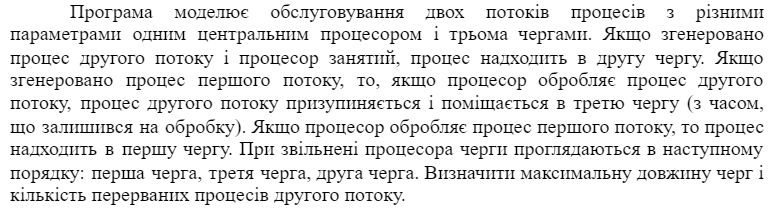
Яременко В. С.

Київ – 2021

# Мета роботи

Метою роботи є набуття навичок роботи з потоками при програмуванні на мові Java або будь-якій іншій.

# Варіант 21



# Програмна реалізація (Thread Tests – main class)

**Main.java**

import java.util.Arrays;  
  
public class Main {  
  
 public static void main(String[] args) throws InterruptedException {  
 int[] processToGenerate = {5, 10};  
 int[] CPUQueuesOrder = {0, 2, 1};  
  
 CPUQueue[] q = new CPUQueue[CPUQueuesOrder.length];  
 for (int i = 0; i < CPUQueuesOrder.length; i++) {  
 q[i] = new CPUQueue(i);  
 }  
  
 CPUQueuesManager M = new CPUQueuesManager(q, CPUQueuesOrder);  
 CPU C = new CPU(M);  
 CPUDispatcher dispatcher = new CPUDispatcher(C, M);  
  
 CPUProcess[] processes = new CPUProcess[processToGenerate.length];  
 for (int i = 0; i < processToGenerate.length; i++) {  
 processes[i] = new CPUProcess(i, processToGenerate[i], dispatcher);  
 }  
 C.start();  
 for (CPUProcess p: processes){  
 p.start();  
 }  
 for (CPUProcess p: processes){  
 p.join();  
 }  
 synchronized (System.*out*){  
 System.*out*.println("-------------------------------------------------");  
 System.*out*.println("CPU was interrupted " + dispatcher.getInterruptsCount() + " times");  
 System.*out*.println("CPUQueues MAX sizes " + Arrays.*toString*(M.getMaxSizes()));  
 System.*out*.println("-------------------------------------------------");  
 }  
 }  
}

**CPU.java**

class CPU extends Thread{  
 Process workProcess;  
 CPUQueuesManager queues;  
  
 CPU(CPUQueuesManager queues){  
 this.queues = queues;  
 }  
  
 public void run(){  
 while(true) {  
 long workTime = 0;  
 if (workProcess == null){  
 try{  
 workProcess = queues.getProcess();  
 System.*out*.println("CPU got a process " + workProcess.getId() + " and i will work " + workProcess.getRunTime() + "\n");  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 long startTime = System.*currentTimeMillis*();  
 try {  
 Thread.*sleep*(workProcess.getRunTime());  
 } catch (InterruptedException e) {  
 workTime = System.*currentTimeMillis*() - startTime;  
 System.*out*.println("CPU was interrupted while serving process" + workProcess.getId());  
 if (workTime < workProcess.getRunTime()){  
 workProcess.setRunTime(workProcess.getRunTime() - workTime);  
 try {  
 queues.setProcess(workProcess, 2);  
 } catch (InterruptedException interruptedException) {  
 interruptedException.printStackTrace();  
 }  
 }  
 } finally {  
 System.*out*.println("I finished a process" + workProcess.getId());  
 workProcess = null;  
 try {  
 queues.throwWorkProcess();  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
 }  
}

**CPUDispatcher.java**

public class CPUDispatcher {  
  
 CPU cpu;  
 CPUQueuesManager queues;  
 int interruptsCount = 0;  
  
 CPUDispatcher(CPU cpu, CPUQueuesManager CPUQueues){  
 this.cpu = cpu;  
 this.queues = CPUQueues;  
 }  
  
 public void dispatchProcess(Process process){  
 Process wP = null;  
 try {  
 wP = queues.getWorkProcess();  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 switch (process.getPpid()){  
 case 0:  
 if (wP == null){  
 System.*out*.println("CPU is free");  
  
 try {  
 queues.setProcess(process, 0);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 break;  
 }else {  
 System.*out*.println("CPU is serving process " + wP.getPpid());  
 if (wP.getPpid() == 1){  
 System.*out*.println("--------------INTERRUPT------------------");  
 interruptsCount++;  
 cpu.interrupt();  
 }  
 try {  
 queues.setProcess(process, 0);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 break;  
 case 1:  
 if (wP == null){  
 System.*out*.println("CPU is free");  
  
 try {  
 queues.setProcess(process, 1);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 break;  
 }else {  
 System.*out*.println("CPU is serving process " + wP.getPpid());  
 try {  
 queues.setProcess(process, 1);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 break;  
 }  
 }  
  
 public int getInterruptsCount() {  
 return interruptsCount;  
 }  
}

**CPUQueuesManager.java**

public class CPUQueuesManager {  
 private final CPUQueue[] queue;  
 private final int[] order;  
 Process workProcess;  
 Boolean isCpuFree = true;  
  
 CPUQueuesManager(CPUQueue[] queue, int[] order){  
 this.queue = queue;  
 this.order = order;  
 }  
 public synchronized void setProcess(Process process, int queueID) throws InterruptedException{  
 queue[queueID].put(process);  
 }  
 public Process getProcess() throws InterruptedException{  
 Process p = null;  
 while (p == null){  
 for(int i: order){  
 p = queue[i].get();  
 if (p != null){  
 break;  
 }  
 }  
 }  
 synchronized (isCpuFree){  
 workProcess = p;  
 isCpuFree = false;  
 }  
 return p;  
 }  
 public Process getWorkProcess() throws InterruptedException {  
 synchronized (isCpuFree){  
 return workProcess;  
 }  
 }  
 public void throwWorkProcess() throws InterruptedException {  
 synchronized (isCpuFree){  
 workProcess = null;  
 isCpuFree = true;  
 }  
 }  
 public int[] getMaxSizes(){  
 int[] size = new int[queue.length];  
 for (int i = 0; i < queue.length; i++) {  
 size[i] = queue[i].getMaxSize();  
 }  
 return size;  
 }  
}

**CPUQueue.java**

import java.util.LinkedList;  
import java.util.Queue;  
  
class CPUQueue {  
  
 private Queue<Process> queue = new LinkedList<>();  
 private final int id;  
 private int maxSize = 0;  
  
 public CPUQueue(int id) {  
 this.id = id;  
 }  
 public synchronized void put(Process p) throws InterruptedException {  
 queue.add(p);  
 if(queue.size()>maxSize){  
 maxSize=queue.size();  
 }  
 System.*out*.println("Process " + p.getId() + " added to queue[" + id + "], queue size = [" + queue.size() + "]");  
 }  
  
 public synchronized Process get() throws InterruptedException {  
 if (queue.isEmpty()){  
 return null;  
 }else {  
 Process item = queue.remove();  
 System.*out*.println("Process " + item.getId() + " removed from queue [" + id + "], queue size = [" + queue.size() + "]");  
 return item;  
 }  
 }  
  
 public synchronized int getId() {  
 return id;  
 }  
  
 public synchronized int getMaxSize() {  
 return maxSize;  
 }  
}

**CPUProcess.java**

class CPUProcess extends Thread{  
  
 int generateNumber;  
 int pid;  
 CPUDispatcher dispatcher;  
  
 CPUProcess(int pid, int gN, CPUDispatcher cD){  
 this.pid = pid;  
 this.generateNumber = gN;  
 this.dispatcher = cD;  
 }  
 public void run(){  
 long generateDelay;  
 for (int i = 0; i < generateNumber; i++) {  
 int randMin=10;  
 int randMax=40;  
 generateDelay = randMin + (int) (Math.*random*() \* randMax);  
 try {  
 Thread.*sleep*(generateDelay);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 Process process = new Process(i, pid);  
 System.*out*.println("Process " + process.getId() + " generated with delay " + generateDelay + "ms");  
 dispatcher.dispatchProcess(process);  
 }  
  
 }  
}

**Process.java**

import java.util.Arrays;  
  
public class Process{  
  
 int pid;  
 int ppid;  
 long runTime;  
  
 Process(int pid, int ppid){  
 this.pid = pid;  
 this.ppid = ppid;  
 int randMin=20;  
 int randMax=80;  
 this.runTime = randMin + (int) (Math.*random*() \* randMax);  
 }  
 public long getRunTime(){  
 return runTime;  
 }  
 public void setRunTime(long rT){  
 runTime = rT;  
 }  
 public String getId(){  
 int[] id = {ppid, pid};  
 return Arrays.*toString*(id);  
 }  
 public int getPpid(){  
 return ppid;  
 }  
}

# Результати роботи програми та їх правильність

-------------------------------------------------

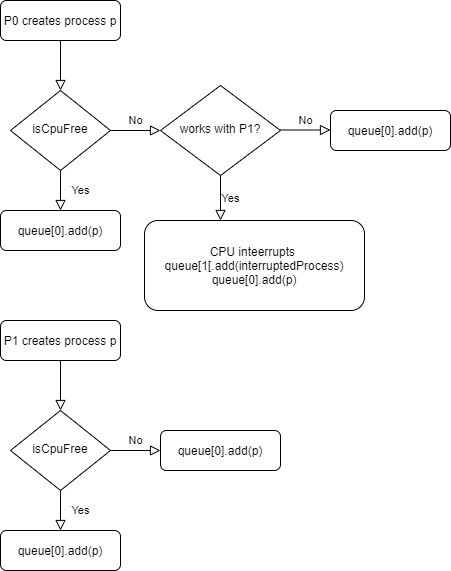
CPU was interrupted 1 times

CPUQueues MAX sizes [4, 9, 1]

-------------------------------------------------

Вище зображено результат роботи програми з таким вхідним масивом int[] processToGenerate = {5, 10};

# Інтерпретація рішення у вигляді схем

Одразу скажу, що в умові не було сказано до яких черг додавати процесі у випадку коли процесор вільний, тому я вирішив, що вони будуть йти у черги індексі яких співпадають з індексами генеруючих потоків.

**Висновки**

У даній роботі я розробив програму, що реалізує варіант моєї задачі, виконавши всі умови.