Algoritmi di PCAD

Thread produttore consumatore

condivisi:

```
type item = ... ;
var buffer: array [0..n-1] of item;
in, out, counter := 0;
```

produttore

```
while (true)
produce un item in nextp
        while (counter = n) {}
         buffer[in] := nextp;
         in := in + 1 mod n;
        counter := counter + 1;
}
```

consumatore

```
while (true) {
        while (counter = 0) { }
        nextc := buffer[out];
        out := out + 1 mod n;
        counter := counter − 1;
}
```

PSC

mutua esclusione , progresso , attesa limitata

se p2 lento allora p1 deve easpettare

Alternanaza stretta → non rispetta la nozione di progresso

condivisa:

var turn (0 o 1) = 0

algoritmo:

```
while (TRUE) {
        while (turn != i) {}
         //sezione critica
        turn := j;
        //sezione non critica
}
```

Peterson per 2 Thread

condivisa:

```
var turn:int;
var interested0,interested1: bool;
```

Process P0:

```
while (TRUE) {
        interested0:=true;
        turn:=0;
        while (turn == 0 and interested1) {}
         //critical section
        interested0:=false;
}
```

Process P1

```
while (TRUE) {
        interested1:=true;
        turn:=1;
        while (turn == 1 and interested0) {}
        //critical section
        interested1:=false;
}
```

Lamport Per N thread

condivisa

```
var choosing: array[1,. . . ,N] of boolean;
 var number: array[1,. . . ,N] of integer;
```

```
Process P(i):
while (TRUE) {
        choosing[i] = TRUE;
        number[i] = max{number[1], . . . , number[N]} + 1;
        choosing[i] = FALSE;
        for k : 1 to N {
                while choosing[k] {}
                while (number[k] != 0 and (number[k], k<<  number[i], i))
                }
        }
        //critical section
        number[i] = 0;
}
```

Fine PSC

Lock Free Programming

algoritmi con istruzioni speciali di test and set

TS(x,oldx) := atomico(oldx := x ; x := 1)

TS per mutua esclusione :

condivisa

```
shared var lock=0;

process P:
bool old;
          while (true){
          while(!old) TS(lock, old);
 //critical section
 lock:=0;
 //non-critical section
}
```

CAS per mutua esclusione

CAS(x,y) := atomico(aux:=x; x:=y; y:=aux;)

condivisa

```
shared var lock=0;

process P:
bool old;
while (true) {
           while(!old){
           old=1;
           CAS(lock, old);
          }
//critical section
lock:=0;
//non-critical section
}
```

SEMAFORI Sezione critica per N processi

condivisa:
Variabili condivise:
var mutex : semaphore
inizialmente mutex = 1

Processo Pi

```
while (TRUE) {
        down(mutex);
        sezione critica
        up(mutex);
        //sezione non critica
}
```

Esempio produttore consumatore con Semafori

condivisa:

```
var mutex : semaphore
var empty : semaphore
var full : semaphore
inizializzate a mutex = 1
empty=N % numero di slot
full=0
```

Produttore

```
int item;
while(TRUE) {
        item=produce item();
        down(empty);
        down(mutex);
        insert item(item);
        up(mutex);
        up(full);
}
```

Consumatore

```
int item;
while(TRUE) {
        down(full);
        down(mutex);
        item=remove item();
        up(mutex);
        up(empty);
        consume item(item);
}
```

Monitor produttore consumatore

```
//monitor
ProducerConsumer
        condition full, empty; //
        integer count;
        procedure insert(item: integer);

        begin
                if count = N then wait(full);
                insert_item(item);
                count := count + 1;
                if count = 1 then signal(empty)
        end;

        function remove: integer;
        begin
                if count = 0 then wait(empty);
                remove = remove_item;
                count := count − 1;
                if count = N − 1 then signal(full)
        end;
        count := 0;
end monitor;

//Produttore
Process Producer:
        while (true)
        item:=...
        ProducerConsumer.insert(item);
//Consumatore
Process Consumer:
        while (true)
        item:=ProducerConsumer.remove;
```

Fine LockFreeProgramming

Libreria di Pthread

Esempio GENERALE di produttore consumatore pseudocodice con mutex lock

```
Produttore
        while(true){
                //produci x
```

```
                lock(mtx)
                //sezione critica
                 // inserisci x nel buffer
                signal(C)
                unlock(mtx)
        }
```

Consumatore

```
while(true){
        lock(mtx)
        while (empty) {
                wait(C,mtx);
                //sc:
                // estrai y dal buffer
                unlock(mtx);
                //elabora y
        }
```

Codice C produttore consumatore con mutex Lock

condiviso:

```
struct node {
        int info;
        struct node * next;
}; /* testa lista inizialmente vuota*/

struct node * head=NULL; /* mutex e cond var*/ pthread_mutex_t
mtx=PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t cond=PTHREAD_COND_INITIALIZER;
```

Produttore

```
static void* producer (void*arg) {
        struct node * p;
        for (i=0; i<N; i++) {
                p=produci();
```

```
                pthread_mutex_lock(&mtx);
                inserisci(p);
                pthread_cond_signal(&cond);
                pthread_mutex_unlock(&mtx);
        }
}
```

Consumatore

```
static void* consumer (void*arg) {
        struct node * p;
        while(true) {
                pthread_mutex_lock(&mtx);
                while (head == NULL){
                        pthread_cond_wait(&cond, &mtx);
                        printf("Waken up!\n");
                        fflush(stdout);
                }
        p=estrai();
        pthread_mutex_unlock(&mtx);
        /* elaborazione p … … */
        }
}
```

Pthread Barrier di sincronizzazione

//creazione

```
pthread_barrier_t barrier = PTHREAD_BARRIER_INITIALIZER(count);
```

//attesa da poszionare dove devono aspettare

```
int pthread_barrier_wait(pthread_barrier_t *barrier);
```

JAVA implementations Fixed Threadpool con executor

//main

```
import java.util.concurrent.Executors;
import java.util.concurrent.ExecutorService;
```

```
public class RunnableTester {
	public static void main(String[] args) {
		Task task1 = new Task("thread1");
		Task task2 = new Task("thread2");
		System.out.println("Starting threads");
		ExecutorService threadExecutor=Executors.newFixedThreadPool(2);
		threadExecutor.execute(task1);
		threadExecutor.execute(task2);
		threadExecutor.shutdown();
		System.out.println("Threads started, main ends\n");
	}
}
```

Java produttore consumatore ThreadPool con BlockingQueue

```
//main
public class BlockingQueueExample {
	public static void main(String[] args) throws Exception {
		BlockingQueue queue = new ArrayBlockingQueue(1024);
		Producer producer = new Producer(queue);
		Consumer consumer = new Consumer(queue);
		new Thread(producer).start();
		new Thread(consumer).start();
		Thread.sleep(4000);
	}
}
```

Producer

```
	public class Producer implements Runnable{
		protected BlockingQueue queue = null;
		public Producer(BlockingQueue queue) { this.queue = queue; }
		public void run() {
			try {
```

```
                    queue.put("1");
                    queue.put("2");
                    queue.put("3");
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }
```

Consumatore

```
        public class Consumer implements Runnable {
            protected BlockingQueue queue = null;
            public Consumer(BlockingQueue queue) { this.queue = queue; }

            public void run() {
                try {
                    var item = queue.take();
                    // consuma item
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
```

Blocking Queue: Implementazione custom

BlockingQueue

```
public class BlockingQueue {
        private List queue = new LinkedList();
        private int limit = 10;
        public BlockingQueue(int limit){ this.limit = limit; }
        public synchronized void enqueue(Object item) throws InterruptedException{
            while(this.queue.size() == this.limit) {
                wait();
            }
            if(this.queue.size() == 0) {
                notifyAll();
            }
            this.queue.add(item);
        }
        public synchronized Object dequeue() throws InterruptedException{
            while(this.queue.size() == 0){
                wait();
            }
            if(this.queue.size() == this.limit){
```

```
                notifyAll();
            }
            return this.queue.remove(0);
        }
}
//ThreadPool che usa la blockingQueue
public class ThreadPool {
        private BlockingQueue taskQueue = null;
        private List<PoolThread> threads = new ArrayList<PoolThread>();
        private boolean isStopped = false;

        public ThreadPool(int noOfThreads, int maxNoOfTasks){
            taskQueue = new BlockingQueue(maxNoOfTasks);
            for(int i=0; i<noOfThreads; i++){
                threads.add(new PoolThread(taskQueue));
            }
            for(PoolThread thread : threads){
                thread.start();
            }
        }
        public void synchronized execute(Runnable task){
            if(this.isStopped) throw new IllegalStateException("ThreadPool is stopped");
            this.taskQueue.enqueue(task);
        }
        public synchronized void stop(){
            this.isStopped = true;
            for(PoolThread thread : threads){
                thread.stop();
            }
        }
}
//PoolThread mando in run il task estratto con dequeue

public class PoolThread extends Thread {
        private BlockingQueue taskQueue = null;
        private boolean isStopped = false;
        public PoolThread(BlockingQueue queue){ taskQueue = queue; }

        public void run(){
            while(!isStopped()){
                try{ Runnable runnable = (Runnable) taskQueue.dequeue();
                runnable.run();
            } catch(Exception e){
                //log or otherwise report exception, //but keep pool thread alive.
            }
        }
}
```

Callable

```
// al posto di runnable
import java.util.concurrent.*;
import java.util.concurrent.Callable;

public class MyCallable implements Callable<Integer> {
	public Integer call() throws Exception {
		int result; ... return result;
	}
}
```

Future con callable singolo

```
FutureTask task = new FutureTask (new MyCallable ());
Thread t =new Thread(task);
t.start();
	try{
		int res = task.get(1000L, TimeUnit.MILLISECONDS);
	}
	catch(ExecutionException e) { }
	catch(TimeoutException e) {
		System.out.println("tempo scaduto");
	} catch(InterruptedException e){ … }
```

Future con callable con Executor

```
FutureTask task = new FutureTask (new MyCallable ());
ExecutorService es =Executors.newSingleThreadExecutor ();

es.submit (task);
	try {
		int result = task.get ();
		System.out.println ("Result from task.get () = " + result);
	} catch (Exception e) {
		System.err.println (e);
	}
es.shutdown ();
```

Monitor Java (synchronized)

esempio completo monitor java

```
class PrenotazioneSicura {
        private int posti=20;
        public int get() { return posti; }
        public synchronized boolean prenota(int richiesta) {
                if (posti >= richiesta) {
                        posti -= richiesta; return true;
                } return false;
        }
 }

public class Cliente extends Thread {
        private Prenotazione p;
        private int richiesta;
        public Cliente(Prenotazione X,int r){
```

```
                this.p=X; this.richiesta=r;
        }
        public void run(){
                if (p.prenota(this.richiesta))
                        System.out.println("ok: "+richiesta);
        } ...
}

public static void main(String[] args) throws InterruptedException {
        PrenotazioneSicura P = new PrenotazioneSicura();
        // oggetto condiviso
        Cliente t1=new Cliente(P,3);
        Cliente t2=new Cliente(P,10);
        Cliente t3=new Cliente(P,5);
        Cliente t4=new Cliente(P,2);
        Cliente t5=new Cliente(P,4);
        Cliente t6=new Cliente(P,2);
        t1.start();
        t2.start(); t3.start(); t4.start(); t5.start(); t6.start();
        System.out.println("Posti disponibili: "+P.get());
 }


//esempio di reentrant sync
public class MyClass {
        public synchronized int foo (int x) {
                if (x>0) return x+foo(x-1);
                else return 0;
        }
        public static void main(String arg[]) {
                MyClass obj=new MyClass();
                System.out.println("risultato: "+obj.foo(4));
        }
}

//esempio di  synchronized ereditato
class base {
        public void  synchronized m1(){
                //qualcosa
        }
}

class derivata extends base {
        public void  m1(){
                //chiamata bloccante al metodo del padre
                super.m1()
        }
}
```

esempio produttore consumatore con monitor

```
//Produttore
class Produttore extends Thread {
	Cella cella;
	public Produttore (Cella cella)		{
		this.cella = cella;
	 }
	public void run() {
		for (int i = 1; i <= 10; ++i) {
		synchronized (cella) {
			while (!cella.scrivibile) {
				try {
					cella.wait();
				} catch (InterruptedException e) {
					return;
				}
			}
		++(cella.valore);
```

```
                cella.scrivibile = false; // cede il turno
                cella.notify(); // risveglia il consumatore }
                }
        }
}

//Consumatore

class Consumatore extends Thread {
        Cella cella;
        public Consumatore (Cella cella)      {
                this.cella = cella;
         }
        public void run() {
                int valore letto;
                for (int i = 0; i < 10; ++i) {
                synchronized (cella) {
                        while (cella.scrivibile) {
                                try {
                                        cella.wait();
                                }catch (InterruptedException e) {
                                        return;
                                }
                        }
                valore letto = cella.valore;
                cella.scrivibile = true; // cede il turno
                cella.notify(); // notifica il produttore
                }
        }
}
}// fine Monitor
Confinamento per thread


// classe di esempio di isolamento di thread
public class ThreadLocalExample {
        public static class MyRunnable implements Runnable {
                private ThreadLocal<Integer> threadLocal = new ThreadLocal<Integer>();
        @Override
        public void run() {
        threadLocal.set( (int) (Math.random() * 100D) );
                try {
                        Thread.sleep(2000);
                } catch (InterruptedException e) { }
        System.out.println(threadLocal.get());
        }
        }

public static void main(String[] args) {
        MyRunnable sharedRunnableInstance = new MyRunnable();
        Thread thread1 = new Thread(sharedRunnableInstance);
        Thread thread2 = new Thread(sharedRunnableInstance);
```

```
        thread1.start();
        thread2.start();
        thread1.join(); //wait for thread 1 to terminate
        thread2.join(); //wait for thread 2 to terminate
        }
}
```