Relazione Progetto Supermercato

INDICE

Introduzione del lavoro 3

Descrizione consegna 3

Eventi da gestire 3

Suddivisione del lavoro 3

COSTRUZIONE SCHEMA UML CLASSI 4

Schema delle classi 4

Codice 4

Punti critici 10

PROBLEMI 10

Cosa ho imparato ? 11

Introduzione del lavoro

Il lavoro assegnato era quello di creare un' applicazione che simulasse gli eventi in un supermercato utilizzato i Thread e la scrittura in file Json e xml.

Descrizione consegna

Il programma prevede che i prodotti del magazzino vengano immagazzinati dentro un file (xml o json a scelta dello studente). Il supermercato ha tre casse dove i clienti devono pagare e dove i prodotti poi una volta passati in cassa vengono scalati dal magazzino. L'incasso di giornata viene salvato su un file esterno (il contrario del file del magazzino (se prima è stato fatto xml lo si fa in json e viceversa)). Ogni cliente verrà gestito come un Thread, dove ognuno di esso andrà in giro per il supermercato e inserirà nel proprio carrello i prodotti scelti. Una volta finita la spesa, un cliente si recherà in una delle tre casse: se libera procederà al pagamento, se occupata attenderà il completamento dei clienti avanti a lui.

Quando un cliente entra nel supermercato parte il Thread, dove con eventi random che avvengono ogni 2 secondi, prosegue il suo cammino nel supermercato.

Eventi da gestire

1 – Acquista un prodotto se disponibile

2 – Chiedi dove si trova un prodotto da un commesso

3 – Riponi un del tuo carello a posto perché sbagliato

4 – Solo dopo che siano passati 6 secondi, il cliente si dirige in cassa

Il totale dei soldi spesi verrà poi salvato in un file xml chiamato "spesaTotale.xml"

Suddivisione del lavoro

Il lavoro, svolto singolarmente, è stato suddiviso in 4 parti:

1) Costruzione schema UML per le classi
2) Sviluppo software
3) Relazione del lavoro in file Word
4) Presentazione preparata in PowerPoint

COSTRUZIONE SCHEMA UML CLASSI

Per la costruzione dello schema UML è stata utilizzata l'applicazione App Diagrams.

Nel progetto sono state usate due classi la prima è la classe Supermercato in cui vengono svolte le funzioni riguardanti le casse, dei thread, lettura e scrittura file (sia json che xml) e acquisto e rimozione del prodotto dal carello.

Schema delle classi

Codice

```
import xml.etree.ElementTree as ET
import json
import threading
import random
import time
import multiprocessing


#inizializzo i semafori
sem1=multiprocessing.Semaphore(1) #cassa1
sem2=multiprocessing.Semaphore(1) #cassa2
sem3=multiprocessing.Semaphore(1) #cassa3

class Supermarket(threading.Thread):
```

```
    negozio = []
    prodotti_nome = []
    filePathJson =
r"E:\EnricoFioreInformtaica\python\5AI\Progetto_Supermercato\Magazzinov2.json"
    filePathXml =
r"E:\EnricoFioreInformtaica\python\5AI\Progetto_Supermercato\SpesaTotale.xml"

    def __init__(self):
        threading.Thread.__init__(self)  # creo thread


    def Sleep(self,n):
        time.sleep(n)

    def Casse(self):
        #self.AzzeraFileXml()
        #self.ImportaDatiFileJson()
        spesa = self.EventiCasuali(Cliente.GetCosto(self))#faccio fare gli eventi
per 6 secondi prima di entare alla cassa
        Cliente.SetCosto(self,spesa) # lo inserisco nella classe cliente
        while True:
            if(sem1.acquire(block=False)):#se il semaforo è libero
                print("Cassa1 occupata\n")
                #self.SalvaPagamento(Cliente.GetCosto(self))
                print("carello: ", Cliente.GetCarello(self))
                print(Cliente.GetNome(self)," ", Cliente.GetCognome(self)," ha
pagato: ",Cliente.GetCosto(self))
                time.sleep(5) #simulare attesa in cassa
                sem1.release()
                print("Cassa1 libera")
                break

            elif(sem2.acquire(block=False)):#se il semaforo è libero
                print("Cassa2 occupata\n")
                #self.SalvaPagamento(Cliente.GetCosto(self))
                print("carello: ",Cliente.GetCarello(self))
                print(Cliente.GetNome(self)," ", Cliente.GetCognome(self)," ha
pagato: ",Cliente.GetCosto(self))
                time.sleep(5) #simulare attesa in cassa
                sem2.release()
                print("Cassa2 libera")
                break

            elif(sem3.acquire(block=False)):#se il semaforo è libero
                print("Cassa3 occupata\n")
                #self.SalvaPagamento(Cliente.GetCosto(self))
                print("carello: ", Cliente.GetCarello(self))
                print(Cliente.GetNome(self)," ", Cliente.GetCognome(self)," ha
pagato: ",Cliente.GetCosto(self))
                time.sleep(5) #simulare attesa in cassa
```

```
                sem3.release()
                print("Cassa3 libera")
                break



    def AggiungiProdottoAlCarello(self):
        print("sono in aggiungo prodotto\n")
        for prodotto in self.negozio:
            self.prodotti_nome.append(prodotto["nome"]) #salvo i nomi dei prodotti
nella list
        nome_prodotto = random.choice(self.prodotti_nome) #prendo un prodotto
random tra quelli salvati nella list
        for prodotto in self.negozio:
            if nome_prodotto == prodotto["nome"]: #se il prodotto corissponde a
quello scelto
                if(prodotto["quantita"]> 0): # e non è finito
                    prodotto["quantita"] -= 1 # decremento
                    print(Cliente.GetNome(self), " " , Cliente.GetCognome(self)," ha
inserito nel carello: ", nome_prodotto)
                    Cliente.GetCarello(self).append(nome_prodotto) # lo salvo nel
carello
                    return prodotto["costo"]# ritorno il prezzo
                else:
                    print("prodotto esaurito\n")
                    return 0



    def ImportaDatiFileJson(self):
        with open(self.filePathJson, 'r') as f:
            data = json.load(f)
        for prodotto in data:
            self.negozio.append(prodotto)

    def SalvaPagamento(self, prezzo):
        tree = ET.parse(self.filePathXml)  # Apro il file XML
        root = tree.getroot()

        # Trovo l'elemento "totale" nel file XML e aggiungo il prezzo
        totale = root.find("./totali/totale")
        if totale is not None:
            totale.text = str("{:.2f}".format(float(totale.text) + prezzo))
        else:
            print("Elemento 'totale' non trovato.")

        tree.write(self.filePathXml)  # Salvo le modifiche al file XML


    def AzzeraFileXml(self):
```

```
        tree = ET.parse(self.filePathXml)
        root = tree.getroot()
        totale = root.find("./totali/totale")
        totale.text = "0.0"
        tree.write(self.filePathXml)

    def RipongoProdotto(self,spesa_costo):
        print("ripongo prodotto\n")
        if(spesa_costo > 0 and Cliente.GetCarello(self) != []) : # se ho prodotti
nel carello
            nome_prodotto = random.choice(Cliente.GetCarello(self)) # prendo un
prodotto a caso dal carello
            for prodotto in self.negozio: # controllo dove il prodotto è presente
nel negozio
                if nome_prodotto == prodotto["nome"]:
                        prodotto["quantita"] += 1 # incremento la quantità
                        print("ho riposto: ", nome_prodotto)
                        Cliente.GetCarello(self).remove(nome_prodotto)#rimuovo il
prodotto dal carello
                        return prodotto["costo"] # ritorno il prezzo per
decrementarlo dal totale
        else:
            print(Cliente.GetNome(self), " " , Cliente.GetCognome(self), " non ha
prodotti nel carello\n")
            return 0



    def EventiCasuali(self,costo):
        start_time = time.time()
        while time.time() - start_time < 6:
            n_random = random.randint(1, 3)
            if n_random == 1:
                costo += self.AggiungiProdottoAlCarello()
            elif n_random == 2:
                print(Cliente.GetNome(self), Cliente.GetCognome(self), " chiede
dove si trova il latte?")
            elif n_random == 3:
                costo -= self.RipongoProdotto(costo)
            time.sleep(2)  # ogni 2 secondi parte evento
        return costo



    """def run(self):
        print("è entrato nel supermercato")
        self.Casse(self)"""


class Cliente(Supermarket):
```

```
    def __init__(self,nome,cognome):
        self.nome = nome
        self.cognome = cognome
        self.carello = []
        self.costo = 0

        super().__init__()

    def GetNome(self):
        return self.nome

    def GetCognome(self):
        return self.cognome

    def GetCarello(self):
        return self.carello

    def GetCosto(self):
       return float("{:.2f}".format(self.costo)) # formula per arrotondare fino a
0.00

    def SetCosto(self,costo):
        self.costo = costo

    def run(self):
        print("È entrato nel supermercato")
        self.Casse() #faccio partire il thread da qui

carello  = []
cliente1 = Cliente("Mario","Rossi")
cliente2 = Cliente("Luca","Bianchi")
cliente3 = Cliente("Giovanni","Verdi")
cliente4 = Cliente("Paolo","Neri")
cliente5 = Cliente("Enrico","Fiore")
cliente6 = Cliente("Giovanni","Virile")
clinete7 = Cliente("Marco","Pela")
cliente7 = Cliente("Gianmarco","Roberti")
cliente8 = Cliente("John Paul","Magsino")
cliente9 = Cliente("Nicolo","Isotti")
cliente10 = Cliente("Even","Bellucci")
cliente11 = Cliente("Lorenzo","Bottegoni")
cliente12 = Cliente("Kunal","Sharma")
cliente13 = Cliente("Daniele","Riccardo")
cliente14 = Cliente("Nicolo","Vero")
cliente15 = Cliente("Davide","Renzi")
cliente16 = Cliente("Gianmarco","Belardinelli")
cliente17 = Cliente("Alessio","Pesaresi")
cliente18 = Cliente("Alessandro","Roccetti")
cliente19 = Cliente("Mattia","Di Lorenzo")
cliente20 = Cliente("Lorenzo","Bastianelli")
```

```
cliente21 = Cliente("Samuele","Tomori")
cliente22 = Cliente("Francesco","Massimo")
cliente23 = Cliente("Michela","Giampietro")
cliente24 = Cliente("Riccardo","Cotani")
cliente25 = Cliente("Leonardo","Papa")

clienti =
[cliente1,cliente2,cliente3,cliente4,cliente5,cliente6,clinete7,cliente7,cliente8,c
liente9,cliente10,cliente11,cliente12,cliente13,cliente14,cliente15,cliente16,clien
te17,cliente18,cliente19,cliente20,cliente21,cliente22,cliente23,cliente24,cliente2
5]

supermercato1 = Supermarket()

supermercato1.AzzeraFileXml() #azzero il file prima di eseguire i thread
supermercato1.ImportaDatiFileJson() #importo i dati dal file json

for i in clienti:
    i.start()#avvio i thread

for i in clienti:
    i.join()#attendo che finiscano i thread
    supermercato1.SalvaPagamento(i.GetCosto())
```

Punti critici

Durante questo lavoro ho riscontrato parecchi problemi.

I problemi riscontrati sono stati: scrittura e lettura dei file (soprattutto file xml), stampa spesa in due cifre decimali (0.00), funzionamento della coda delle casse, funzionamento del lock per la mutua esclusione dei thread.

PROBLEMI

1) Scrittura e lettura dei file (sia json che xml)
2) stampa spesa in due cifre decimali (0.00)
3) funzionamento della coda delle casse

La scrittura e lettura dei file, era un problema riguardante il file xml, esso riguardavo un avviso di errore che usciva sul terminale.

Tale errore si è risolto dichiarando la funzione di azzeramento e assegnamento della spesa nel file xml, rispettivamente all'inizio e alla fine della partenza dei thread.

La stampa spesa in due cifre decimali (0.00) era un problema riguardante la stampa del costo totale delle spese che andava oltre le due cifre decimali.

Tale problema è stato risolto tramite la formula: "{:.2f}".format(variabile)

Il funzionamento della coda nelle casse era un problema riguardante l'assegnazione della cassa libera ai thread.

Inizialmente avevo provato ad usare il lock per effettuare la mutua esclusione tra i veri thread, ma siccome tale problema persisteva, ho utilizzato il semaforo per gestire la dispnibilità o meno delle casse.

Cosa ho imparato ?

Durante questo progetto svoltosi in solitaria ho imparato ad organizzare il lavoro in modo autonomo e cavarvela da solo nelle situazioni di difficoltà.

Per quanto riguarda il codice ho imparato ad organizzare il lavoro tra Thread usufrendo anche dei semafori, ho imparato anche a scrivere e leggere file sia di tipo Xml e Json.