Tekst projektnog zadatka

U proizvoljnom objektno-orijentisanom programskom jeziku realizovati raspoređivač zadataka. Definisati programski API za raspoređivač i realizovati logički odvojenu GUI aplikaciju za raspoređivanje zadataka koja koristi definisani API. Kreirati tip zadatka koji vrši paralelnu obradu multimedijalnih podataka. Pravilno dokumentovati projekat. Program se mora moći kompajlirati, izvršiti i testirati. Uz priloženi rad treba da se obezbijede i primjeri koji se mogu iskoristiti za testiranje programa. Obezbijediti nekoliko jediničnih testova kojima se demonstriraju funkcionalnosti iz stavki u specifikaciji projektnog zadatka (nije potrebno za GUI aplikaciju i fajl-sistem). Pridržavati se principa objektno-orijentisanog programiranja, SOLID principa, principa pisanja čistog koda i konvencija za korišteni programski jezik.

Pravilno realizovati osnovne funkcionalnosti za raspoređivač zadataka. Raspoređivač zadataka mora biti realizovan u sklopu biblioteke. Obezbijediti jednostavan API koji korisniku omogućava da vrši raspoređivanje zadataka. Omogućiti da se za raspoređivač specificira maksimalan broj zadataka koji se konkurentno izvršavaju. Dozvoljeno je da se definiše sopstveni API koji mora biti korišten u implementaciji zadataka, kao način da se kooperativnim mehanizmima omoguće funkcionalnosti iz specifikacije projektnog zadatka. Dozvoljeno je i da se izvršavanje svakog zadataka pokreće kao zaseban proces. Obezbijediti jednostavne jedinične testove koji demonstriraju rad raspoređivača. Osnovne funkcionalnosti koje API raspoređivača treba da omogući su:

- dodavanje zadatka u red raspoređivača, sa započinjanjem ili bez započinjanja,
- raspoređivanje zadataka za vrijeme dok se izvršavaju već raspoređeni zadaci,
- započinjanje, pauziranje, nastavak i zaustavljanje zadatka,
- čekanje na kraj izvršavanja zadatka i
- specifikacija algoritma za raspoređivanje, što uključuje FIFO (FCFS) i prioritetno raspoređivanje.

Omogućiti da se za zadatak mogu opcionalno specificirati: datum i vrijeme početka izvršavanja, dozvoljeno ukupno vrijeme izvršavanja i rok do kojeg zadatak mora biti izvršen ili prekinut.

Omogućiti da se za svaki zadatak specificira i dozvoljeni nivo paralelizma (npr. dozvoljen broj iskorištenih procesorskih jezgri). Definisati tip zadatka za obradu multimedijalnih fajlova. Konkretan algoritam odabrati na kursu predmeta. U GUI aplikaciji obezbijediti podršku za rad sa zadacima definisanog tipa. Zadatak definisanog tipa treba da se može rasporediti na izvršavanje upotrebom raspoređivača. Zadatak treba omogućiti istovremenu obradu veće količine ulaznih fajlova.

Realizovati zadatak tako da on može da vrši paralelnu obradu i nad samo jednim multimedijalnim fajlom. Pri tome se treba ostvariti ubrzanje (testirati nad multimedijalnim fajlom adekvatne veličine).

Omogućiti da se vrši prioritetno raspoređivanje sa preuzimanjem, kao i raspoređivanje bazirano na vremenskim isječcima, gdje zadaci višeg prioriteta dobijaju više vremenskih isječaka za izvršavanje. Mehanizmima kooperativnog raspoređivanja simulirati preuzimanje.

Omogućiti rad sa resursima, tako da zadaci mogu da zaključaju resurse u toku izvršavanja, pri čemu su resursi identifikovani jedinstvenim stringom. Pri tome, obezbijediti mehanizme za prevenciju ili detekciju i razrješavanje deadlock situacija. Riješiti problem inverzije prioriteta (PIP ili PCP algoritam).

Obezbijediti aplikaciju sa neblokirajućim grafičkim korisničkim interfejsom za raspoređivanje zadataka. Konstruisati aplikaciju tako da se može lako proširiti novim tipovima zadataka. Omogućiti korisniku aplikacije da specificira način raspoređivanja. Omogućiti korisniku aplikacije da specificira sva svojstva kreiranog zadatka (dozvoljeno vrijeme izvršavanja, rokove, itd.). GUI aplikacija treba da pruži podršku za pregled zadataka u toku izvršavanja, uz prijavu progresa zadatka putem progres bara, kao i za kreiranje, započinjanje, pauziranje, nastavljanje, zaustavljanje i uklanjanje završenih ili zaustavljenih zadataka.

Omogućiti serijalizaciju i deserijalizaciju zadataka. Serijalizovana polja mogu da uključuju, kao primjer, niz putanja na ulazne fajlove i putanju na kojoj treba da se generišu izlazni fajlovi. Omogućiti da se interno stanje aplikacije može serijalizovati (broj zadataka i stanje obrade). Omogućiti da se, u slučaju bilo kakvog zaustavljanja aplikacije (uključujući i neočekivano zaustavljanje), zadaci mogu nastaviti izvršavati (npr. autosave mehanizam) od tačke u kojoj su se prethodno zaustavili. Pretpostaviti da su zadaci idempotentni.

Napisati drajver za fajl-sistem u korisničkom prostoru. Fajl-sistem treba da koristi API raspoređivača i da omogući izvršavanje zadatka za obradu multimedijalnih podataka, na sljedeći način:

- Fajl-sistem treba da sadrži folder input u koji se mogu kopirati multimedijalni fajlovi
- Nakon što su fajlovi kopirani u folder input, započinje obrada tih fajlova
- Nakon što završi obrada fajlova, rezultati treba da budu dostupni u folderu output

Text of the Project Task

Implement a task scheduler in an arbitrary object-oriented programming language. Define a scheduler programming API and implement a logically separate task scheduling GUI application that uses the defined API. Create a task type that performs parallel processing of multimedia data. Document the project properly. The program must be able to be compiled, executed and tested. Examples that can be used to test the program should be provided along with the attached work. Provide several unit tests that demonstrate the functionalities from the items in the specification of the project task (not required for the GUI application and the file system). Adhere to the principles of object-oriented programming, SOLID principles, principles of writing clean code and conventions for the used programming language.

Properly implement the basic functionalities for the task scheduler. The task scheduler must be implemented as part of the library. Provide a simple API that allows the user to schedule tasks. Allow the scheduler to specify a maximum number of concurrently executed tasks. It is allowed to define your own API that must be used in the implementation of tasks, as a way to enable cooperative mechanisms with functionalities from the specification of the project task. It is also allowed to run each task as a separate process. Provide simple unit tests that demonstrate the operation of the scheduler. The basic functionalities that the Scheduler API should provide are:

- adding a task to the scheduler queue, with or without starting,
- scheduling tasks while performing already scheduled tasks,
- starting, pausing, resuming and stopping a task,
- waiting for the end of task execution and
- specification of the scheduling algorithm, which includes FIFO (FCFS) and priority scheduling.

Make it possible to optionally specify for the task: the date and time of the start of execution, the allowed total execution time and the deadline by which the task must be executed or terminated.

Make it possible to specify the allowed level of parallelism for each task (eg the allowed number of used processor cores). Define the type of task for processing multimedia files. Choose a specific algorithm in the subject course. In the GUI application, provide support for working with tasks of a defined type. A task of a defined type should be able to be scheduled for execution using a scheduler. The task should enable the simultaneous processing of a large number of input files.

Realize the task so that it can perform parallel processing on only one multimedia file. In doing so, acceleration should be achieved (test on a multimedia file of adequate size).

Enable priority scheduling with takeover, as well as time-slice-based scheduling, where higher-priority tasks get more time-slices to execute. Simulate takeover with cooperative scheduling mechanisms.

Enable resource manipulation, so that tasks can lock resources at runtime, where the resources are identified by a unique string. At the same time, provide mechanisms for the prevention or detection and resolution of deadlock situations. Solve the priority inversion problem (PIP or PCP algorithm).

Provide an application with a non-blocking graphical user interface for task scheduling. Design the application so that it can be easily extended with new types of tasks. Allow the application user to specify the deployment method. Allow the application user to specify all the properties of the created task (allowed execution time, deadlines, etc.). A GUI application should provide support for viewing running tasks, reporting task progress via a progress bar, and for creating, starting, pausing, continuing, stopping, and removing completed or stopped tasks.

Enable serialization and deserialization of tasks. Serialized fields can include, for example, an array of paths to input files and a path where output files should be generated. Allow the internal state of the application to be serializable (number of tasks and processing state). Allow that, in the event of any application termination (including unexpected termination), tasks may continue to run (eg, autosave mechanism) from the point at which they were previously terminated. Assume that assignments are idempotent.

Write a driver for the file system in user space. The file system should use the scheduler API and enable the execution of the multimedia data processing task, as follows:

- The file system should contain an input folder into which multimedia files can be copied
- After the files have been copied to the input folder, the processing of those files begins
- After processing the files, the results should be available in the output folder.

Text der Projektaufgabe

Implementieren Sie einen Aufgabenplaner in einer beliebigen objektorientierten Programmiersprache. Definieren Sie eine Scheduler-Programmier-API und implementieren Sie eine logisch separate GUI-Anwendung für die Aufgabenplanung, die die definierte API verwendet. Erstellen Sie einen Aufgabentyp, der die parallele Verarbeitung von Multimediadaten durchführt. Dokumentieren Sie das Projekt ordnungsgemäß. Das Programm muss kompiliert, ausgeführt und getestet werden können. Beispiele, die zum Testen des Programms verwendet werden können, sollten zusammen mit der beigefügten Arbeit bereitgestellt werden. Stellen Sie mehrere Einheitentests bereit, die die Funktionalitäten der Elemente in der Spezifikation der Projektaufgabe demonstrieren (nicht erforderlich für die GUI-Anwendung und das Dateisystem). Halten Sie sich an die Prinzipien der objektorientierten Programmierung, SOLID-Prinzipien, Prinzipien des Schreibens von sauberem Code und Konventionen für die verwendete Programmiersprache.

Implementieren Sie die grundlegenden Funktionen für den Taskplaner ordnungsgemäß. Der Taskplaner muss als Teil der Bibliothek implementiert werden. Stellen Sie eine einfache API bereit, die es dem Benutzer ermöglicht, Aufgaben zu planen. Ermöglichen Sie dem Planer, eine maximale Anzahl gleichzeitig ausgeführter Aufgaben anzugeben. Es ist erlaubt, eine eigene API zu definieren, die bei der Umsetzung von Aufgaben verwendet werden muss, um kooperative Mechanismen mit Funktionalitäten aus der Spezifikation der Projektaufgabe zu ermöglichen. Es ist auch erlaubt, jede Aufgabe als separaten Prozess auszuführen. Stellen Sie einfache Komponententests bereit, die die Funktionsweise des Planers demonstrieren. Die grundlegenden Funktionen, die die Scheduler-API bereitstellen sollte, sind:

- Hinzufügen einer Aufgabe zur Scheduler-Warteschlange, mit oder ohne Start,
- Zuweisung von Aufgaben während der Ausführung bereits zugewiesener Aufgaben,
- Starten, Pausieren, Fortsetzen und Stoppen einer Aufgabe,
- Warten auf das Ende der Aufgabe und
- Spezifikation des Scheduling-Algorithmus, der FIFO (FCFS) und Prioritäts-Scheduling umfasst.

Möglich machen, für die Aufgabe optional anzugeben: Datum und Uhrzeit des Ausführungsbeginns, zulässige Gesamtausführungszeit und Frist, bis zu der die Aufgabe ausgeführt oder beendet werden muss.

Ermöglichen Sie die Angabe des zulässigen Parallelitätsgrades für jede Aufgabe (z. B. die zulässige Anzahl verwendeter Prozessorkerne). Definieren Sie den Aufgabentyp für die Verarbeitung von Multimediadateien. Wählen Sie im Fachkurs einen bestimmten Algorithmus aus. Bieten Sie in der GUI-Anwendung Unterstützung für die Arbeit mit Aufgaben eines definierten Typs. Ein Task eines definierten Typs sollte mit einem Scheduler zur Ausführung eingeplant werden können. Der Task soll die gleichzeitige Verarbeitung einer großen Anzahl von Eingabedateien ermöglichen.

Realisieren Sie die Aufgabe so, dass sie nur eine Multimediadatei parallel verarbeiten kann. Dabei sollte eine Beschleunigung erreicht werden (Test an einer Multimediadatei ausreichender Größe).

Aktivieren Sie Prioritätsplanung mit Übernahme sowie zeitscheibenbasierte Planung, bei der Aufgaben mit höherer Priorität mehr Zeitscheiben für die Ausführung erhalten. Simulieren Sie Übernahmen mit kooperativen Scheduling-Mechanismen.

Aktivieren Sie die Ressourcenmanipulation, damit Tasks Ressourcen zur Laufzeit sperren können, wobei die Ressourcen durch eine eindeutige Zeichenfolge identifiziert werden. Stellen Sie gleichzeitig Mechanismen zur Verhinderung oder Erkennung und Auflösung von Deadlock-Situationen bereit. Lösen Sie das Prioritätsinversionsproblem (PIP- oder PCP-Algorithmus).

Stellen Sie eine Anwendung mit einer nicht blockierenden grafischen Benutzeroberfläche für die Aufgabenplanung bereit. Gestalten Sie die Anwendung so, dass sie einfach um neue Arten von Aufgaben erweitert werden kann. Erlauben Sie dem Anwendungsbenutzer, die Bereitstellungsmethode anzugeben. Ermöglichen Sie dem Anwendungsbenutzer, alle Eigenschaften der erstellten Aufgabe anzugeben (zulässige Ausführungszeit, Fristen usw.). Eine GUI-Anwendung sollte das Anzeigen laufender Aufgaben, das Melden des Aufgabenfortschritts über einen Fortschrittsbalken und das Erstellen, Starten, Anhalten, Fortsetzen, Stoppen und Entfernen abgeschlossener oder gestoppter Aufgaben unterstützen.

Aktivieren Sie die Serialisierung und Deserialisierung von Aufgaben. Serialisierte Felder können beispielsweise ein Array von Pfaden zu Eingabedateien und einen Pfad enthalten, in dem Ausgabedateien generiert werden sollen. Lassen Sie zu, dass der interne Zustand der Anwendung serialisierbar ist (Anzahl der Tasks und Verarbeitungsstatus). Zulassen, dass im Falle einer Anwendungsbeendigung (einschließlich einer unerwarteten Beendigung) Tasks ab dem Punkt, an dem sie zuvor beendet wurden, weiter ausgeführt werden können (z. B. Autosave-Mechanismus). Gehen Sie davon aus, dass Zuweisungen idempotent sind.

Schreiben Sie einen Treiber für das Dateisystem im Userspace. Das Dateisystem sollte die Scheduler-API verwenden und die Ausführung der Multimedia-Datenverarbeitungsaufgabe wie folgt ermöglichen:

- Das Dateisystem sollte einen Eingabeordner enthalten, in den Multimediadateien kopiert werden können
- Nachdem die Dateien in den Eingabeordner kopiert wurden, beginnt die Verarbeitung dieser Dateien
- Nach der Verarbeitung der Dateien sollten die Ergebnisse im Ausgabeordner verfügbar sein