Gabriele **Proietti Mattia**

Advanced Operating Systems and Virtualization

[Lab 06] Final Project

DIAG
Department of Computer, Control and Management Engineering "A. Ruberti", Sapienza University of Rome

gpm.name · proiettimattia@diag.uniroma1.it

A.Y. 2020/2021 · v1

General Rules

This year's project regards the implementation of a **User Mode Thread Scheduling (UMS)** system. The feature is currently available in the Windows kernel, therefore the specification of the project has been taken from the official windows documentation available here.

The project should be developed **individually** or in a group of **2 people**. You have **1 year** for carrying out this project (up to February 2022), otherwise it will change.

The project must be developed in the private repo of **GitHub Classroom** by using Git.

You can ask me clarifications about the track, and I can also give suggestions about where to look in the kernel source but you **do not need to abuse of my availability** and I cannot provide you code snippets.

The project track is even available on the course site, under "News" or "Exam" sections.

Specification

You are requested to implement a **User Mode thread Scheduling** (UMS) facility which makes able the programmer of a C user space application to **schedule threads** without involving the kernel scheduler. The ability to switch threads in user mode makes UMS more efficient than thread pools for managing large numbers of short-duration work items that require few system calls.

The application which is intended to use UMS needs to implement a **UMS scheduler** component which is in charge to determine which is the thread to be scheduled. The creation of the scheduler receives as input:

- an ***entrypoint***, that is a function that is executed for determining the next thread to be scheduled and therefore it is called when a the scheduled is started the first time, when a work thread ends, yields (with `UmsThreadYield`, see next slide) or is put on wait (e.g. page fault);
- a set of worker threads, that we also call a ***completion list***.

Specification

Multiple UMS schedulers can share threads or completion lists (so be careful for race conditions), and a UMS Scheduler has always associated a ***scheduler thread*** (in general an application creates a scheduler thread for each core/processor available in the system). The scheduler thread will **host** the execution of the worker threads.

You, at least, have to implement:

- `EnterUmsSchedulingMode` which **converts** a standard **pthread** in a **UMS Scheduler thread**, the function takes as input a completion list of worker threads and a entry point function;
- `DequeueUmsCompletionListItems` called from the scheduler thread obtains a **list** of current **available threads** to be run, if no thread is available to be run the function should be blocking until a thread becomes available;
- `ExecuteUmsThread` called from a scheduler thread, it **executes** the passed **worker thread** by switching the entire context;
- `UmsThreadYield` called from a worker thread, it **pauses** the execution of the current thread and the UMS scheduler entry point is executed for determining the next thread to be scheduled;

Specification

The backend implementation is up to you, you can also decide to build a worker thread ready queue (as in Windows specification) in order to speed up the lookup of ready threads. In that case, implement also the `GetNextUmsListItem` that adds a worker thread from the list returned by `DequeueUmsCompletionListItems` to the ready queue of the scheduler.

You are also required to implement:

- expose in `` `/proc/ums/<pid>/schedulers` `` a file for each scheduler thread (named with numbers from 0 to n) that shows some statistics as the number of times the scheduler switched to a worker thread, the completion list, the time needed for the last worker thread switch, the current state (idle/running) and if it is running, which worker thread in the completion list;
- expose in `` `/proc/ums/<pid>/schedulers/<id>/workers` `` a file for each worker thread (named with numbers from 0 to n as in the sorting order of the completion list) that shows some statistics, as the total running time of the thread, the number of switches, the current state (idle/running)

You can and need to implement any other function you think that is useful for adhering to track.

Specification

Every notion, detail, particular behaviour, parameter or variable that is not mentioned in this track **is left to you** to decide and if you have doubt you can take inspiration from the original Windows specification

https://docs.microsoft.com/en-us/windows/win32/procthread/user-mode-scheduling

You **do not need** to be fully compatible with the Windows implementation, this track is lightened with respect the original. Moreover, you **do not need to be retro-compatible** with previous kernel versions, choose a single version of the kernel and develop on it. If your code works even with older versions it's better but it is not requested.

What to hand in

You need to produce the following:

1. the **kernel implementation** of the project, that can be a kernel module or you can just modify the kernel source (both for any kernel version that you desire from 2.4 to 5.12). In any case you need to push the source on your private repo of **GitHub classroom**;
2. a small **C library** with a proper header .h which allows to use your implemented features;
3. an **example code** which shows that your implementation works with **more than one** scheduler thread and with UMS schedulers which shares worker threads;
4. the **documentation** of the code. You are encouraged to use an automation tool for generating the documentation, like Doxygen or similar and put the output in **/doc** as PDF.
5. a **short final** report describing the project decisions and some benchmark of your solution, as the average time needed for a scheduler thread to switch a worker thread or any kind of parameter you want to measure, follow the template in the **/doc** folder

What will be evaluated

The project mark has a weight of ⅖ on the final mark and will be evaluated in /30. The evaluation criteria will take into account the following characteristics of your project (in order of importance):

1. adherence to the project **track** regarding the implemented features;
2. **correctness** of the solution;
3. **questions** about the project (asked individually) during the project presentation (you obviously need to know the theory behind the facilities that you use);
4. elegance, modularity and cleanliness of the **code** (you are encouraged to follow how the kernel source is written, e.g. use macros for optimizing things and not functions);
5. clarity and quality of the **documentation**;
6. final **report** and results;
7. good usage of **git** and versioning.

What to do now

0. obviously be joined to the course Google Classroom;
1. create a **GitHub account**, adding your `@studenti.uniroma1.it` email to the list of your addresses
2. accept this **invitation link**
3. if requested please **associate** your name with one in the class
4. now create the **Team**:
   a. if you **will work alone** create a team with your **name and surname**, well written, like "Name Surname", there is a space between name and surname;
   b. if you **will work in group** (max 2) create a team with the **two surnames**, like "Surname1 Surname2", there is a space between the 2 surnames.

   I trust in your common sense for not adding people that you do not know. Moreover, please **be 100% sure** of your decision before creating the team.
5. start to **reason** about the project but keep in mind that several topics that will be useful will be presented later in the course

Suggestions

Some suggestion to do the project:

- reason and design the solution before writing code
- use a good kernel source explorer
- use a virtual machine with your custom kernel version and do not perform updates
- use Visual Studio Code
- use the "Test Assignment" of GitHub classroom for playing with the repository, link here

Pull Requests

Pull requests for feedback are enabled for your project repository, they are described well here. In practice you can create a pull request for asking clarifications to me but **you cannot abuse of this feature** and my answers are obviously limited, I cannot provide you code snippets and I cannot do the project for you. The answers to general questions will be made available to everyone in the project track page as F.A.Qs.

Advanced Operating Systems and Virtualization

[Lab 06] Final Project

LECTURER

Gabriele **Proietti Mattia**

gpm.name · proiettimattia@diag.uniroma1.it

DIAG