Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

System Construction Course 2015,

Assignment 7

Felix Friedrich, ETH Zürich, 27.10.2015

Implementation of Priority Inheritance

Lessons to Learn

- Understand the problem of Priority Inverion and how to deal with it
- Get deeper insight into scheduling and process synchronisation algorithms of $\mathcal{A}_2$

Preceding Remark

- We are considering the A2 System with 3 user priorities (Low, Normal, High).
- The *development system* is WinAos or Linux Aos, an emulation of A2 under Windows/Linux. The scheduling part of WinAos/LinuxAos is implemented using Windows/Linux threads and thus does not strictly follow the scheduling protocol of A2.
- Henceforth as *target system* we will use A2 running in a virtual machine. This A2 system does *not yet* comprise priority inversion handling.
- Install VirtualBox on your device. If you feel uncomfortable with this, you can use any virtualisation environment that emulates modern x86 hardware and that has a support for booting from raw disk images as IDE devices.
- We have prepared a script to build A2 for execution in VirtualBox (or VMWare). The virtual disk `A2HDD.img` is zipped as `A2HDD.zip`. Please extract this file before working on the tasks.

Preparation

The first task can be executed on the *target system* completely. You do not need to start A2 on your host.

Task 1

Compile and execute program `TestPrioInv.Mod` on the target system. Analyse its behavior by looking at the source code of `TestPrioInv.Mod` and the log output. What implication has the scheduling policy?

Task 2

Modify procedure `Lock` of module `Objects.Mod` in such a way that it produces some console output if priorities are supposed to be inherited. Use module `Log.Mod` for logging, and `LogWindow.Open` to open a window containing the output produced via `Log.Mod`. We are using this special logging facility in order to prevent the system from ample logging during booting. Use `TestPrioInv.Mod` to trigger the console output.

Task 3

Implement the priority inversion handling protocol that was presented in the lecture.

- First make sure that priorities are set and reset correctly in procedures `Lock` and `Unlock`. In a second step the priority propagation should be implemented as a helper procedure `PropagatePrio`. Add respective calls in `Lock` and `Unlock`.
  Hint: No additional fields in the data structure are required. The counter fields are already present at the data structure of processes `Objects.Process.prioRequests` and object headers `Heaps.ProtRecBlockDesc.waitingPriorities`
- Update procedures `Await` and `TransferLock` accordingly.

Important Hint: Do not change the external interface of the Module Objects, as you would need to recompile the whole system otherwise!

Documents

- System Construction Lecture 7 slides from the course-homepage
  `http://lec.inf.ethz.ch/syscon`
- $\mathcal{A}_2$ Programming Quickstart Guide. File A2QuickStartGuide.pdf in folder `documents/oberon`