Process Scheduling

Pedro H. Penna, Henrique Freitas,
Márcio Castro and Jean-François Méhaut

Pontifical Catholic University of Minas Gerais
Federal University of Santa Catarina
University of Grenoble Aples

Abstract

Processes are they key abstraction of every operating system, in which in fact the entire system is built upon. In this assignment, you will work with the main module of the process management subsystem in Nanvix, the process scheduler. First, you shall study the current implementation, and then you shall propose enhancements in it.

Background

Nanvix is a multitasking operating system. That is, it supports concurrent execution of multiple programs. To deliver this feature, likewise in other operating systems, Nanvix relies on processes, an abstraction of a running program that encapsulates the execution flow, variables, stack and all other important information that concerns the program itself. With this abstraction, all that the system does is to switch back and forth between the processes, by reloading machine registers and environment variables with the data from the process that is being admitted to run. This way, the operating system provides the illusion that programs are running simultaneously.

However, an important step relies on the selection of which process to run. With many processes in the system, there are several possibilities: run the process that is waiting longer, run the process with the shortest remaining time next, or alternatively run the process that has the highest priority. This choice is actually carried out by the scheduler component of the process management subsystem and it has great impact on the performance, responsiveness and fairness of the system.

Assignment Description

The process scheduling in Nanvix follows the simple round-robin policy: it assigns to each process a *quantum* of time, and then it schedules them in a first-in first-out fashion. For doing so, the operating system maintains for each process a counter, which keeps track of the time in which the process is waiting to be executed. When the *quantum* of the running process ends, the process that is waiting longer is reassigned a new quantum and than it is selected to run.

This policy is easy to implement and it delivers fairness to single-task systems. However, it is not enough for more robust systems that support multitasking. For instance, imagine a scenario in which there is an interactive process, like the terminal, and other 99 cpu-hungry processes. If the process scheduler assigns 100 ms of quantum to each process, each process runs every ten seconds. Although this may be fine for the 99 process, it is surely not for the interactive process – the user definitely would not enjoy his/her experience.

TO address the above problem, you should implement a fairer scheduling policy, which is based on priorities. In this alternative approach, at every scheduling decision, the process with the highest priority should be selected for running. This priority-based scheme may be static, but keep in mind that a dynamic-priority mechanism may be interesting. With this assignment you should deliver a brief report about your solution. In addition, we will provide benchmarking programs so that you ca test and argue about the strengths and weaknesses of you design.

Starting Point

The implementation of the process management subsystem is in the `kernel/pm` directory, and it is split into several files:

- `pm.c`: initializes the process management subsystem.
- `sched.c`: process scheduler.
- `sleep.c`: implementations for `sleep()` e `wakeup()`.

In this assignment, you should focus on the `sched.c`. More precisely in the `yield()` routine, which is the implementation of the process scheduler itself.