Konzepte von Betriebssystem-Komponenten

Linux Scheduler

Valderine Kom Kenmegne
Valderinek@hotmail.com

Contents:

1. Introduction

In our daily life we have a lot of tasks for example in our family, at work, at school … therefore we have to think about how we can manage our time so that we won't be overloaded. To resolve this problem we can use an agenda. But it remains some situation where we cannot execute all our tasks or there are many tasks at time, in this cases we try to order our tasks in priorities depending on our interests and goals. Our operating system (OS) knows also about this problem, the tasks to be executed will be called this time **processes.** Like a man does not have more than 24 hours per days to execute his task, an OS has only a limited number of processors for many processes. To manage this situation, our OS has to choose a scheduling algorithm.

In the first part of our study we will focus on the goals of scheduling and scheduling algorithms in OS in general, later we will define our OS as Linux and base our study on Linux 2.4.

2. Scheduler policy in Operating Systems

In this section we will get some knowledge about the objectives of scheduling in OS and about some scheduling strategies.

2.1 Scheduling Objectives

During the design of the scheduler, the designer has to consider several factors like the application area of the OS and the need of potential users . Depending on these aspects the scheduler can have as purpose:

- *Maximize the number of interactive processes receiving acceptable response time.*
- *Maximize resource utilization*. The scheduling mechanism should keep the resources of the system busy.
- *Avoid indefinite postponement.* A process should not experience an unbounded wait time before or while receiving service.
- *Minimize overhead.* Interestingly, this is not generally considered to be one of the most important objectives. Overhead often results in wasted resources. But a certain portion of system resources effectively invested as overhead can greatly improve overall system performance.
- *Ensure predictability.* By minimizing the statistical variance in process response time, a system can guarantee that processes will receive predictable service levels.

Despite the differences in the goals amount systems, many scheduling algorithm exhibit similar properties:

Fairness. A scheduling algorithm is fair if all similar processes are treated the same and no process can suffer indefinite postponement due to scheduling issues
Predictability. See above.
Scalability. System performance should degrade gracefully it should not immediately collapse under heavy loads. [1]

To reach these objectives, an OS has to choose a corresponding algorithm. Therefore we will introduce some scheduling algorithm in this part.

2.2 Some Scheduling Strategies

There are two classes of scheduling algorithms: **Non-preemptive:** a process *cannot* be removed from the processor while computing. **Preemptive:** It is the opposite: A process *can* be removed from the processor while computing. Depending on these classes we have different scheduling strategies:

- **First-comes-first-served (FCFS):**

The processes are dispatched according to their arrival time at the ready queues. Once a process has been assigned to a processor, the process runs until completion. This is a *non-preemptive scheduling algorithms.* The disadvantage of FCFS is that short processes can starve while waiting for the processor.

- **Round-Robin (RR):**

RR is similar to FCFS but the processes are given a limited amount of processor time called a **time slice** or a **quantum.** If a process does not complete before its quantum expires, the system preempts it and gives the processor to the next waiting process. The system places the preempted process at the end of the ready queue.

- **Shortest-process-first (SPF):**

Is a non preemptive scheduling algorithm in which the scheduler selects the waiting process with the smallest estimated run-time-to-completion. SPF reduces average waiting time over FCFS. The great problem is to predict the process duration.

- **Shortest-Remaining-Time (SRT):**

Is the preemptive counterpart of SPF. In SPF, once a process begins executing, it runs until its completion. In SRT a newly arriving process with a shorter estimated run-time-to-completion, preempts a running process with a longer run-time-to-completion. Like SPF this scheduling requires also a knowledge about the process duration.

3. Linux Scheduler

In Linux each process has a priority, used by the scheduler to determine the process execution. The process with the greatest priority will receive the processor at first. This priority can be **static:** once a process is given a priority it won't be changed, or **dynamic:** the scheduler adjusts the process priority periodically.
In this third part, let us get some information about Linux processes in general, before focusing on Linux 2.4.

3.1 Different process type in Linux

Linux has three process types: **Interactive**, **batch** and **real-time**. In the following part, we will describe each process type.

- **Interactive processes:**

These processes communicate with the user, therefore, they spend much time waiting on key presses and mouse clicks. They also have to react quickly to an input in average 50 - 150 ms, otherwise the user will find that the system is too slow or unresponsive. The typical interactive processes are: command shells, text editor and graphical application.

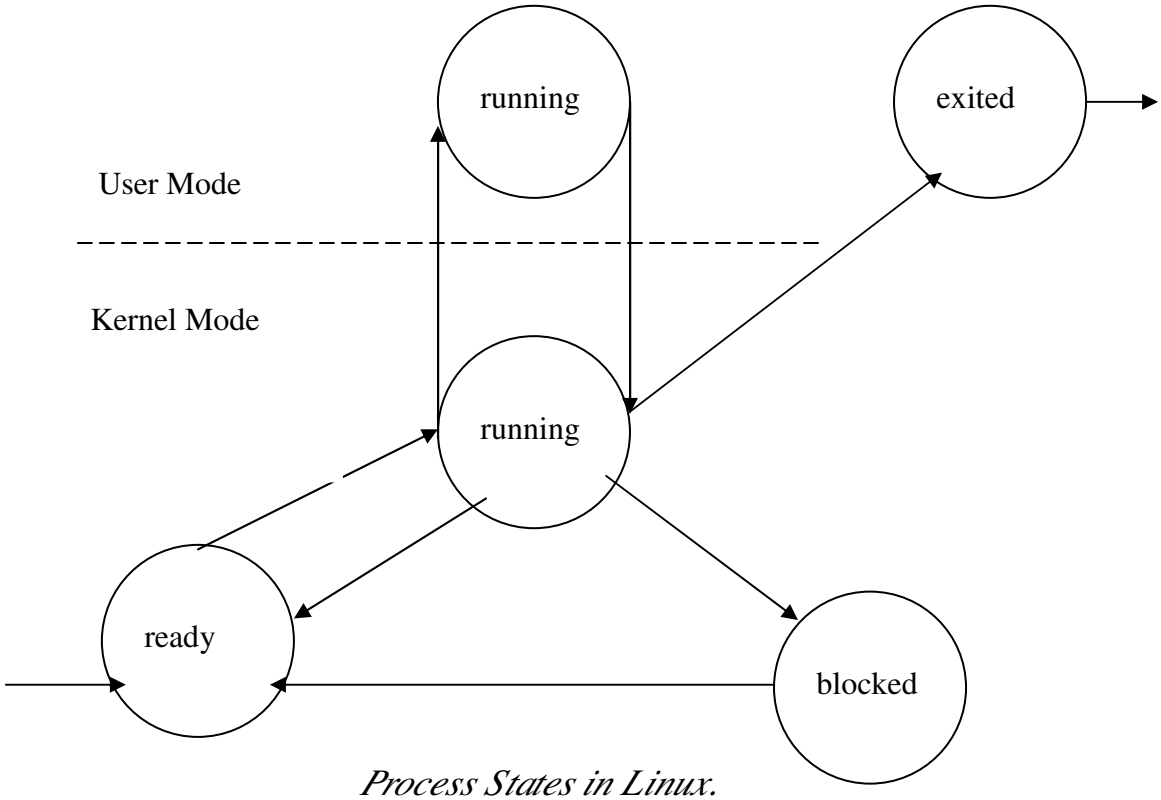
- **Batch processes:**

They do not need to interact with the user and normally run in the background. These processes are often penalized by the scheduler because they don't need to be very responsive. Typical batch processes are programming language compilers, database search machines and scientific computations.

- **Real-time processes:**

These have very strong scheduling of scheduling. Such processes would never be blocked by processes of lower priority, they should have a short response time, such a time of answer should have a minimal discord. Typical real-time programs are video and sound applications, robot controllers, and programs that collect data from physical sensor. [6]

3.2 Process States



Process States in Linux.

Ready state: These processes are waiting to receive the processor.
Blocked state: When a process blocks because it cannot be continued for example when the process waits for user input.
Running state: The process is executed at the moment. Linux makes a difference between the running process in Kernel Mode (these can directly access the hardware) and running in User Mode (for normal user processes).
Exited state: The process has terminated its execution.

3.3 Process Preemption

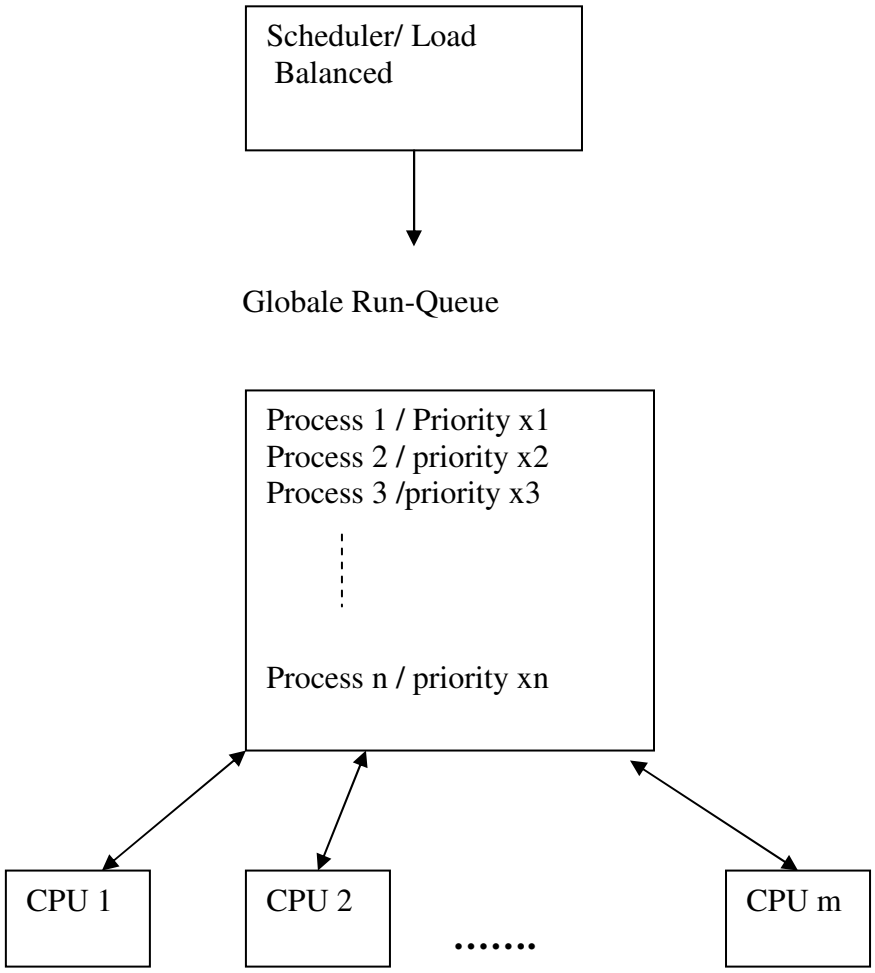
After looking at these process states we can easily find out when a process preemption can occur**,** for example when one of the following event happens:

- *The birth of a new process caused by the system call fork():* The scheduler has to choose between the father and the son process. If the return value of fork() is 0 the son is the running process, if greater than 0 the running process is the father process .
- *The running process blocks.*
- *The reserved time expires.*
- *The running process terminates.*

Now that we have enough basic knowledge about the goal of scheduling, scheduling strategies, process types, process states and process preemption. We can focus our attention on scheduling policy on Linux 2.4.

3.4 Case study: Linux 2.4 scheduling policy.

Every new process receive a priority and goodness value at the beginning and enter if runnable in the run-queue. We will explain later how this goodness is computed. We can already affirm at this stage that Linux 2.4 differentiates **real-time** processes from **conventional processes**. Depending on the process type Linux has to choose a strategy.



Kernel 2.4

3.4.1 Real-time processes

Every process of this type receives a goodness value equal or greater than 1000 and the scheduler uses one of the following scheduling classes: SHED_FIFO or SCHED_OTHER.

- SCHED_FIFO:
This class implements the FCFS algorithm the processes receive a static priority between 0-99. When a Process has the processor it will run as long as it wishes even if other real-time processes having the same priority are runnable. The process will only free the CPU if it blocks, exits or there is another real-time process with a greater priority.
- *SCHED_ RR:*
Is similar to SCHED_FIFO, but each process receives a time slice. See Round Robin strategy in the section 2.2.

3.4.2 Conventional processes

For all other processes SCHED_OTHER is used. A process will start to run if there is no real-time process runnable. These processes can be preempted, their priority is dynamic and can be influenced by calling *nice( )* the negative value can only be given by the super user because if the priority value is higher the priority is smaller. Their goodness value ranges between 1 and 999.

3.4.3 Goodness computation

When choosing the new process to be executed, the scheduler computes for each runnable candidate the goodness value, the one with the greatest goodness value will be chosen. The following table explains this computation.

Scheduling - Strategy	Remaining quantum	Goodness
SCHED_FIFO, RR	-	1000 + priority
SCHED_OTHER	>0	Quantum + priority(+1)
SCHED_OTHER	0	0

By this table, one can already figure out that interactive processes attain a higher goodness value in Linux. They block often while waiting for user input and therefore they consume only a part of their time slice.

3.4.4 Linux 2.4 Disadvantages

-Linux 2.4 has a great overhead due to the run-queue. For each process preemption the scheduler has to compare and compute the goodness of all processes in the queue.
-The kernel has a global run-queue for all runnable processes. It can also happen that a CPU is locked because it waits for another CPU to release this list.

5. Summary

At the beginning we wanted to know how an Operating System in general and Linux in particular determines the process to be executed. To reach this objectives we studied the scheduling algorithms. In Linux the scheduler differentiates between real-time processes and conventional processes. According to this difference it chooses between SCHED_FIFO, SCHED_RR, SCHED_OTHER. In Linux 2.4 the scheduler computes for each process in the run-queue the goodness value, depending on this value, it chooses the process to be executed. However we also saw that this method has a great overhead due to this computation.

6 References

[1] Operating System, Deitel Deitel Choffnes
Third edition, Pearson Education," 2004
[2] Betriebsystem, Casten Vogt
1.Auflage Spektrum, Akademischer verlag, 2001
[3] Moderne Betriebsystem, Andrew s.Tanenbaum
2.Auflage Pearson Studium
[4] Übung zu Softwaresystem I, Jürgen Kleinöder, 2005

[5] IX, 2/2005 (Zeitschrift)

[6] Understanding the Linux Kernel, Daniel P. Bovet, Marco Cesati
O'Reilly, 2001