3

TASKS: Management of Tasks (a.k.a. Processes)

3.1 Chapter Objective

The objective of the TASKS project is to teach students about task management in a modern-day operating system and to provide them with a well-structured programming environment in which to implement task-management techniques. To this end, students will be asked to implement the OSP 2 class TaskCB, the only class of package TASKS. TaskCB stands for Task Control Block, the OSP 2 object used to represent tasks.

3.2 Conceptual Background

Like other modern operating systems, OSP 2 distinguishes between program execution and resource ownership. The former is captured through the concept of a thread, which represents a running program, and the latter is captured using the concept of a task. In older operating systems, like traditional Unix, the process filled both of these roles; actually, we sometimes use the term “process” as a synonym for task. In OSP 2, a task serves as a “container” for one or more threads, all executing the same code and sharing the same memory address space. Also associated with a task is a swap file containing an image of
the task’s address space, other files opened by the task’s constituent threads, and the communication ports created by these threads. We say that these resources (memory, ports, files, etc.) are owned by the task and shared by the task’s threads; this explains how the issue of resource ownership is organized around the concept of a task.

Threads are the schedulable and dispatchable units of execution in OSP 2. They are sometimes referred to as “lightweight processes” for it is much easier in a multiprogramming OS to switch the CPU from one thread to another than from one process to another, due to above-explained separation of program execution and resource ownership in an OS supporting the task/thread doctrine. We will have more to say about threads in the next chapter.

A task can be created or destroyed, newly created threads can be added to a task, and threads are deleted from the owner task’s thread list after they are destroyed. There is also a system-wide notion of the current task, which is the task that owns the currently running thread. This thread is known as the current thread of the task.

In the rest of this chapter we describe TaskCB, the only class in the Tasks package. The class diagram of Figure 3.1 puts TaskCB in context with related classes.

3.3 Class TaskCB

Tasks are represented by the class TaskCB, which is the only class to be implemented in the Tasks project. It is defined as follows:

◦ public class TaskCB extends If1TaskCB

The following methods are to be implemented as part of this project:

◦ public static void init()
  This method is called at the very beginning of simulation and can be used to initialize static variables of the class, if necessary.

◦ public static TaskCB do_create()
  This method creates a new task object and then initializes it properly.

In OSP 2, creation of a task involves the creation of a task object, allocation of resources to the task, and various initializations. The task object is created using the default task constructor TaskCB(). First, a page table must be created using the PageTable() constructor, and associated with the task using the method setPageTable(). Second, a task must keep track of its threads (objects of type ThreadCB), communication ports (objects of type PortCB),