Operating Systems - Study
EDAF35

Elias Bergström
`el8025be-s@student.lu.se`

26 december 2024

1 Module 2 - Processes and Threads

1.1 Red Box

- It is important to understand what the PCB is and that the PCBs get put on different queues by the OS when managing process state. (From section 3.1.3 to the end of 3.2.1 in Operating System Concepts)

- You need to understand process creation (including fork() and exec() in detail) and process termination (including zombie and orphan processes). From section 3.1.3 to the end of 3.2.1 in Operating System Concepts)

- You need to understand the difference between user threads and kernel threads and the different models for mapping between the two (Section 4.3 in Operating System Concepts)

1.2 Notes

The process can be in a number of different states, see figure 1.
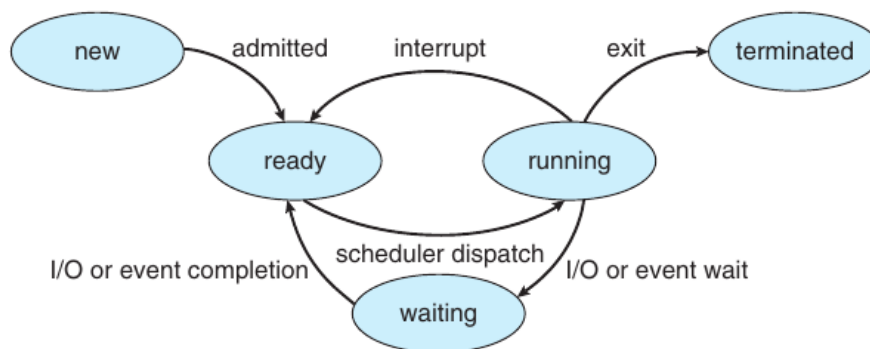
Figur 1: Diagram of process states

1.2.1 PCB

PCB stands for Process Control Block and each process is represented in the OS by one. The PCB contains information about the process:

- **Process state.** The state may be new, ready, running, waiting, halted, and so on.

- **Program counter.** The counter indicates the address of the next instruction to be executed for this process.

- **CPU registers.**

- **CPU-scheduling information.** This information includes a process prior- ity, pointers to scheduling queues, and any other scheduling parameters. (Chapter 5 describes process schedu- ling.)

- **Memory-management information.** This information may include such items as the value of the base and limit registers and the page tables, or the segment tables, depending on the memory system used by the operating system (Chapter 9).

- **Accounting information** This information includes the amount of CPU and real time used, time limits, account numbers, job or process numbers, and so on.

- **I/O status information.** This information includes the list of I/O devices allocated to the process, a list of open files, and so on.

In brief, the PCB simply serves as the repository for all the data needed to start, or restart, a process, along with some accounting data.

fork() and exec() in Unix/Linux Context

In a Unix/Linux context, `fork()` and `exec()` are system calls used for process creation and management.

fork()

The `fork()` system call creates a new process by duplicating the calling (parent) process. The new process is called the *child* process. Both the parent and the child process continue executing from the point of the `fork()` call. The child process gets a copy of the parent's memory space, but they have different Process IDs (PIDs). `fork()` returns:

- `0` in the child process.

- The child's PID in the parent process.

exec()

The `exec()` family of functions (e.g., `execvp()`, `execp()`, etc.) replaces the current process's memory space with a new program. After calling `exec()`, the process image is completely replaced, and the new program starts executing. This is commonly used after `fork()` when the child process needs to run a different program than the parent.

Typical Usage

In typical usage:

1. `fork()` is used to create a new process.

2. `exec()` is used by the child (or parent) to replace its process image with a different program.

Together, these calls enable the creation of new processes and the execution of different programs, which is fundamental for tasks like launching new applications or running shell commands.

1.2.2   Multithreading Models

These models describes how to map user thread to kernel threads. User threads are supported above the kernel and are managed without kernel support and the kernel threads are managed by the kernel.

- **Many-to-One Model**, all user threads are mapped to one kernel thread, where the switching between threads are done by a thread library in user space (not by the kernel) It's efficient but if the current user threads hangs it will also hang the kernel thread.

- **One-to-One Model**, maps each user thread to a kernel thread. Multiple threads can run at the same time. The problem with this model is that you need to create a kernel thread for each user thread.

- **Many-to-Many**, multiplexs many user threads to a smaller or equal amount of kernel threads.

- **Two-Level Model** mixing two of the models.

2 Module 3.A - CPU Scheduling

2.1 Red Box

- Make sure you understand what it means for scheduling to be pre-emptive. (From section 5.1.3 in Operating System Concepts)

- You need to know and understand the tradeoffs between these different algorithms (Section 5.3 in Operating System Concepts)

- Be able to understand the differences between process and system contention scopes (Section 5.4 in Operating System Concepts)

- You need to understand ready queues, load balancing and processor affinity in multiprocessor systems (Section 5.5.1, 5.5.3 and 5.5.4 ins Operating System Concepts)

- You need to understand what makes real time scheduling different, the periodic process model and the differences between the RMS and the EDF scheduler (Section 5.6.1 – 5.6.4 in Operating System Concepts)

3 Module 3.B - Synchronization

3.1 Red Box

- You should know what the critical section problem is (Section 6.2 in Operating System Concepts)

- You must know the differences between Spinlocks, Semaphores and Mutexes in the context of Operating (Systems. 6.5 and 6.6 in Operating System Concepts – not the clearest explanation. Chapter 9, 10 in Linux Kernel development)

3.2 Module 4 - Memory Management

3.2.1 Red Box

- YOU NEED TO UNDERSTAND WHAT THE PHYSICAL AND LOGIC ADDRESS SPACE IS AND THE MOTIVATION BEHIND IT. YOU ALSO NEED TO UNDERSTAND THAT THE MMU IS REQUIRED TO TRANSLATE BETWEEN THE TWO (9.1.1 TO 9.1.4 IN OPERATING SYSTEM CONCEPTS

- YOU NEED TO UNDERSTAND WHAT CONTIGOUS ALLOCATION IS, HOW IT WORKS AND WHY FRAGMENTATION IS A MAJOR ISSUE. (SECTION 9.2 IN OPERATING SYSTEM CONCEPTS)

- YOU NEED TO UNDERSTAND WHAT PAGING IS, WHAT THE PAGE TABLE AND TLB ARE, AND HAVE A GENERAL IDEA OF WHAT PROTECTION AND SHARED PAGES ARE (SECTION 9.2 IN OPERATING SYSTEM CONCEPTS

- YOU NEED TO KNOW WHY WE CANNOT USE SIMPLE PAGE TABLES, THE THREE ALTERNATIVE PAGE TABLE STRUCTURES AND THE ADVANTAGES AND DISADVANTAGES OF EACH (SECTION 9.4 OF OPERATING SYSTEM CONCEPTS

- YOU NEED TO UNDERSTAND THE BASIC CONCEPTS OF VIRTUAL MEMORY AND ITS ADVANTAGES(SECTION 10.1 OF OPERATING SYSTEM CONCEPTS)

- YOU NEED TO BE ABLE TO EXPLAIN WHAT DEMAND PAGING IS, FREE FRAME LIST AND ITS PERFORMANCE (SECTION 10.2 OF OPERATING SYSTEM CONCEPTS

- YOU NEED TO KNOW WHAT PAGE REPLACEMENT IS, UNDERSTAND THE MAIN THREE DIFFERENT PAGE REPLACEMENT ALGORITHMS DISCUSSED IN THE BOOK AND THEIR TRADEOFFS, AND BE ABLE TO DESCRIBE BELADY'S ANOMALY (SECTION 10.4 OF OPERATING SYSTEM CONCEPTS

3.3 Module 4 pt 2 - Memory Management Additional Slides

3.3.1 Red Box

- YOU NEED TO KNOW THE CHALLENGES WITH ALLOCATING MEMORY IN THE OPERATING SYSTEM AND THE DIFFERENCES BETWEEN THE SLAB AND THE BUDDY ALLOCATER (SECTION 10.8 OF OPERATING SYSTEM CONCEPTS)

3.4 Module 6 - File System

3.4.1 Red Box

- You should understand these two different access methods (13.2.1 and 13.2.2 in Operating System Concepts)

- You should understand what directories are and how they make it possible to organise and access files (13.3 in Operating System Concepts)

- This structure is discussed in the textbook but not very clearly. Try and understand it, but we will not ask questions discussing it directly (Section 14.1 in Operating System Concepts).

- Need to that know that files are represented as blocks and what the FCB/Inode is (Section 14.1 in Operating System Concepts).

- Need to understand what these two tables are and how calls like open() and read() use and update this table (Section 14.2.2 in Operating System Concepts)

- Understand these three different allocation methods and their relative advantages and disadvantages (Section 14.4 in Operating System Concepts)

- Be able to calculate maximum file size a scheme like this can store (Section 14.4.3 in Operating System Concepts)

- Understand these two algorithms and their advantages and disadvantages (Section 14.5.1 and 14.5.2 in Operating System Concepts)

3.5   Module 6 - I/O Systems

3.5.1   Red Box

- Need to be able to know what memory mapped I/O is and the motivation for why we use. (Section 12.2.1 in Operating System Concepts)

- You need to understand how these three different methods work and the motivations for each. (Section 12.2.2, 12.2.3 and 12.2.4 in Operating System Concepts)

- You need to understand the differences and needs for these different interfaces. (Section 12.3.1 to 12.3.4 in Operating System Concepts)

- Make sure you understand this flow as it encompasses most of what we have spoken about today. (Section 12.5 in Operating System Concepts)

3.6   Module 7 - Protection and Security

3.6.1   Red Box!

- You should be able to describe what a domain of protection is and give examples of some different domains and objects (Section 17.4 in Operating System Concepts)

- You should be able to describe what the access matrix is, how it relates to domains of protection and how it can be implemented – you will not be asked about the lock and key mechanism, and only on the basics of capability lists (Section 17.5 and 17.6 in Operating System Concepts)

- Maintaining system security is very complicated and understanding this could require a whole courses – you will not be asked on this in the exam

3.7   Module 8 - Virtualisation and Virtual Machines

3.7.1   Red Box

- You should know what virtualisation means and be able to briefly describe a few types of virtualisation eg: VMs, Virtual Networks, Virtual Disks and Virtual Memory

- You need to know the difference between different types of VMs: Type 1 and 2 hypervisors (not type 0 hypervisors), Emulation and Containers. (Section 18.5.3, 18.5.4, 18.5.7 and 18.5.8 in Operating System Concepts)