Operating Systems

Written by HADIOUCHE Azouaou.

Disclaimer

Man… Not sure if there is a course, but it follows what is supposed to be in it. Presented by Mr. METROUH. For the purposes of a better understanding of the course, a small library of operations and functions is made to run the programs that will be given in the course. A proper implementation will be found later.

> To separate the contents of the course to actual additions or out of context information, a black band will be added by its side like the globing this comment.

Contents

Chapter 1

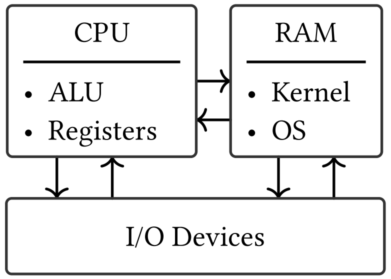
Operating Systems

Operating System is an abstraction layer in software that goes between the user and hardware, which gives the ability to easily access the resources and the services given from the hardware layer to the software.

Definition 1.1 (Operating System): *The OS is a set of essential programs for managing the services of a computer, which interfaces between the user and hardware and facilitates the execution of programs, those are some services*

- ***Basics**: file management, user management, networking…*
- ***Resource Management**: loading files, distributing processes on cores…*
- ***Program Execution**: loading program in RAM and running…*
- ***User Interface**: displaying graphical or command-line interfaces…*

The usual architecture we use in this case is the Von Neumann architecture to represent the computer workarounds which can be represented in a really simple way as follows.



1.1. Historical Evolution Of OSs

1. **First Generation (1950s):** The first operating systems were batch systems where the programs were processed in batches which used punch cards to enter data.
2. **Second Generation (1960s):** introduction of multiple users to share the same computing resources which inccreased productivity.
3. **Third Generation (1970s):** systems start to run interactive tasks simultaneously thus making it more efficent in tasks and the minicomputers made operating systems more accessible and affordable for small scale usages.
4. **Fourth Generation (1980-1990s):** the popularization of micro-computers and personal-computers gave more access to a wide audience and the development of networks and management of shared resources.
5. **Fifth Generation (After 2000s):** The evolution of rebost, secure and cross-platform operating systems that give the best user experience and more user-friendly services and the birth of mobile computers and laptops for portability.

Chapter 2

Process Management

Process management is the part of the operating system that is responsible for the creation, execution, synchronization, and termination of processes. The goal is to have the best algorithms for sharing and managing the resources of the computer to maximize efficiency.

2.1. Basic Concepts

Definition 2.1.1 (Process): *A process is an instance or a program in execution including the binaries or scripts of the running program, also the program data like variables, heap and the execution state information like the contents of the CPU registers, the call stack.*

The process can have the following states

New	the process is in the creation phase, the OS is allocating the resources and initializes the process.
Ready	once the 'New' phase is done, the process becomes ready to execute and waits for the CPU allocation, i.e. it is put in a queue to receive CPU time.
Running	the process is in the top of the queue and is being executed by the CPU.
Waiting/Blocked	the process is waiting for an event like a key press and it will not process until the event happens.
Terminated/Exited	the process has completed its execution, all the allocated resources will be freed.
Suspended/Swapped	the process is moved temporarily from the RAM to another memory to free some RAM for other processes.

Definition 2.1.2 (Process Control Block): *A process control block (PCB) is a data structure that the operating system relies on to manage processes, it contains all the information for controlling the process. For each process, we have a PCB that has the following main components*
- ***PID**: a unique identifier integer assigned for the process.*
- ***Process State**: the state of the process.*
- ***Program Counter**: the address of the next instruction in memory.*
- ***CPU Registers**: the value of registers in the process.*
- ***Stack Pointer**: the address of the process stack.*
- ***Scheduling Information:** process priority, scheduling policy…*
- ***Memory Information:** process's address space, page tables…*
- ***Accounting Information:** resources used by the process.*
- ***Signal Information:** pending signals for the process…*

The PCB is unique for each process and helps resume the process even after an interruption by a change of context. This part is important for multitasking.

2.2. Process Scheduling

scheduling processes aims to optimize usage of the resources by having the following objectives:
- Maximize CPU Usage
- Minimize Waiting & Response Time
- Uniform Distribution Of CPU Time

We will check some standard algorithms for process scheduling.

2.2.1. Process Scheduling Algorithms

In the scheduling information of the PCB, we have some parameters that we will use, in each algorithm we have extra information that we will give and how they are used in the algorithm.

Algorithm 2.1 (FCFS: First-Come, First-Server):

- ***Workflow:*** *the processes are executed in their order in the queue.*
- ***Advantages:***
  - *Easy to implement.*
  - *Fastest execution time.*
- ***Disadvantages:***
  - *Short processes may take long to execute.*
  - *Processes blocking others from execution.*
- ***Implementation:***

```
type Process
  - id: identifier
  - arrival_time: the arrival time
  - burst_time: time needed for the process
  - done: is the process done

algorithm FCFS
  input:
    - processes: process queue

  sort(processes, arrive_time)
  current_time = 0

  for process in processes do
    if current_time < process.arrival_time
      current_time = process.arrival_time
    end
  end
```

Algorithm 2.2 (SJN: Shortest Job Next.):

- ***Workflow*** *the processes are sorted by burst time from shortest to longest and then executed in this order.*
- ***Pros***
  - *minimizes time between arrival and completion.*
  - *short processes are done quicker.*
- ***Cons***
  - *slower execution by the sorting.*
  - *more complicated to implement.*
  - *starvation of longer processes.*

```
type Process
  - id: identifier
  - arrival_time: the arrival time
  - burst_time: time needed for the process
  - done: is the process done

algorithm SJF
  input:
    - processes: process queue

  sort(processes, burst_time)
  while not all(processes, done)
    - get the list of processes not done.
    - execute the first process until its done.
  end
```

Algorithm 2.3 (Priority Scheduling):

- ***Workflow:*** *the processes are sorted by process priority from highest to lowest and then executed in this order.*
- ***Advantages:***
  - *prioritizes critical processes.*
  - *allows control on processes.*
- ***Disadvantages:***
  - *slower execution by the sorting.*
  - *starvation of processes with low priority.*

```
type Process
  - id: identifier
  - arrival_time: the arrival time
  - burst_time: time needed for the process
  - priority: priority of the process
  - done: is the process done

algorithm PriorityScheduling
  input:
    - processes: process queue

  sort(processes, priority)
  while not all(processes, done)
    - get the list of processes not done.
    - execute the first process until its done.
  end
```

Algorithm 2.4 (RR: Round Robin):

- ***Workflow:*** *each process is given a quantum in CPU time and processes are executed in rotation.*
- ***Advantages:***
  - *easy implementation.*
  - *improved response time.*
  - *fairness in distribution of resources to process.*
- ***Disadvantages:***
  - *efficency depends on quantum size.*

```
type Process
  - id: identifier
  - arrival_time: the arrival time
  - burst_time: time needed for the process
  - done: is the process done

algorithm PriorityScheduling
  input:
    - processes: process queue

  sort(processes, priority)
  while not all(processes, done)
    - get the list of processes not done.
    - execute the first process until its done.
  end
```

1. **Multilevel Queue Scheduling**

Information	• *type:* a type to categorize processes.
Workflow	processes are grouped by their type and each group can have its own scheduling algorithm.
Pros	• better control on processes by their types. • multiple algorithms can improve management.
Cons	• queue management can be more complicated. • efficency is highly dependent on the types and the algorithms used for each type.

2. **Multilevel Feedback Queue**

Workflow	Similar to the multilevel queue, but processes may switch their queue depending on their behavior.
Pros	• more flexible by adapting to process behavior. • multiple algorithms can improve management.
Cons	• complicated implementation. • efficency is highly dependent on the types and the algorithms used for each category and extra parameters.

2.3. Communication & Synchronization

Processes need to use common resources or communicate throughout their execution, thus we have multiple synchronization techniques for there not to be any overlap of usage or corruption of resources in use of those processes.

2.3.1. Processes Communication

1. **Pipes:** provide a uni-directional communication channel between processes, usually between a parent process and its children processes.

2. **Message Queues:** allows the exchange of message between processes through a messages queue.
3. **Shared Memory:** multiple processes can access the same region of memory like files and allows a fast and efficient communication.
4. **Sockets:** a socket enables communication over a network using some protocols for exchanging data such as TCP/IP or UDP.

2.3.2. Processes Synchronization

1. **Mutex-Locks**: