FoSAP Lecture Notes

Emma van Emelen

June 2, 2020

Contents

1 Structure and Tasks

1.1 The structure of computer systems

1.2 The Structure and Tasks of an OS

1.3 Multitasking & Interrupts

2 C Programming

3 Shell Programming

4 Process Management

4.1 The Definition of a Process

A process is simply a running program. This program consists of sequentially executed code, which defines state changes. Part of a process is a full description of the CPUs current state, like the program counter, registers, the stack, the heap and any variables or filesthat are being worked on A process is created like this:

1. Create the Process evironment in RAM.
    - create the heap
    - create the stack
    - load program code
    - load libraries
2. wait for CPU assignment
3. Load operation into CPU register and execute
4. If a process is interrupted, its state is saved and another process will get CPU time.

4.2 Memory Allocation

A process gets 3GiB of memory by default. Memory allocation is broken up into 5 parts.

- The stack, which holds function parameters and local variables. This is automatically managed memory.
- The heap, which is dynamically allocated memory `malloc`. This gives the programmer full control over the Memory management but can cause alot of problems, if not used carefully. Heap-Buffer-Overflows are very common when manually controlling the memory allocation. This happens when a program writes more data into memory than the variable has been assigned. This then overwrites any memory that comes after the variable's memory space.
- Statically allocated memory and global data. this is broken up into bss, for **NULL**–initialized variables and data, for other variables.
- The code, which holde the executable program code

4.3 Process Types

1. `The User Process`, for System Programs and Applications. These all run in `User-Mode`
2. `The System Process`, forparts of the Kernel. This includes but is not limited to memory management, file system management and device management. These Processes have a higher priority and operate in `Kernel-Mode`.

There are multiple active processes at any given time, which have to be coordinated efficiently, because they can all use the same resources. These recources may include memory space, CPU time and IO-devices, such as storage, sensors, screens and peripherals.

5 CPU Scheduling

5.1 Tasks and simple strategies

These strategies typically don't apply to modern systems anymore, but make it easier to understand the principles od scheduling.

5.1.1 Tasks of Scheduling

During its lifetime, a process goes into queues, where a scheduler has to decide which process gets the resource. The task of the scheduler is to manage resources for many different processes, dynamically moving a process in and out of queues, as needed. To execute a process, two things need to happen:

1. The process has to be loaded into main memory
2. the process has to get CPU time

There are multiple strategies for scheduling processes.

`Long-Term Scheduling` - This decides which process gets added to the ready queue and can take from minutes to even several days. One example for this would be Backup timing, where it is unnecessary and even inefficient to execute in too little time intervals, as a the process itself can take several hours.

`Mid-Term Scheduling` - This scheduler decides which process is to be executed next and thus will be loaded into memory before execution. This usually takes no more than a few seconds.

`Short-Term Scheduling` - This scheduling happens when we have to decide, which process can use the CPU next. This is done within a few milliseconds.

5.2 Strategies for modern systems

5.3 Multiprocessor/Multicore systems

5.4 Scheduling in Linux