Review and Conclusions

What have you learned? What else could you still learn?

Adrian Coleșa

Technical University of Cluj-Napoca (UTCN)
Computer Science Department

June 5, 2019

The purpose of this chapter

1. review subjects presented during the OS course (this semester)
2. draw some conclusions
3. presents future OS-related courses

The purpose of this chapter

1. review subjects presented during the OS course (this semester)
2. draw some conclusions
3. presents future OS-related courses

The purpose of this chapter

1. review subjects presented during the OS course (this semester)
2. draw some conclusions
3. presents future OS-related courses

Outline

1. OS Subject Review
2. Conclusions
   - Past
   - Future

Outline

1. OS Subject Review

2. Conclusions
   - Past
   - Future

OS Definition, Role, Architecture

- system software placed between hardware and user software (applications)
- roles
  - virtual machine provider (hide hardware and provide abstractions)
  - resource manager
- every resource access / need must be required from OS
  - by calling SO services, i.e. system calls
- OS protects itself from user applications based on different CPU execution modes
  - privileged: kernel mode
  - non-privileged: user mode
- architectures
  - monolith
  - micro-kernel

OS Definition, Role, Architecture

- system software placed between hardware and user software (applications)
- roles
  - virtual machine provider (hide hardware and provide abstractions)
  - resource manager
- every resource access / need must be required from OS
  - by calling SO services, i.e. system calls
- OS protects itself from user applications based on different CPU execution modes
  - privileged: kernel mode
  - non-privileged: user mode
- architectures
  - monolith
  - micro-kernel

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

OS Definition, Role, Architecture

- system software placed between hardware and user software (applications)
- roles
  - virtual machine provider (hide hardware and provide abstractions)
  - resource manager
- every resource access / need must be required from OS
  - by calling SO services, i.e. system calls
- OS protects itself from user applications based on different CPU execution modes
  - privileged: kernel mode
  - non-privileged: user mode
- architectures
  - monolith
  - micro-kernel

OS Definition, Role, Architecture

- system software placed between hardware and user software (applications)
- roles
  - virtual machine provider (hide hardware and provide abstractions)
  - resource manager
- every resource access / need must be required from OS
  - by calling SO services, i.e. system calls
- OS protects itself from user applications based on different CPU execution modes
  - privileged: kernel mode
  - non-privileged: user mode
- architectures
  - monolith
  - micro-kernel

OS Definition, Role, Architecture

- system software placed between hardware and user software (applications)
- roles
  - virtual machine provider (hide hardware and provide abstractions)
  - resource manager
- every resource access / need must be required from OS
  - by calling SO services, i.e. system calls
- OS protects itself from user applications based on different CPU execution modes
  - privileged: kernel mode
  - non-privileged: user mode
- architectures
  - monolith
  - micro-kernel

OS Definition, Role, Architecture

- system software placed between hardware and user software (applications)
- roles
  - virtual machine provider (hide hardware and provide abstractions)
  - resource manager
- every resource access / need must be required from OS
  - by calling SO services, i.e. system calls
- OS protects itself from user applications based on different CPU execution modes
  - privileged: kernel mode
  - non-privileged: user mode
- architectures
  - monolith
  - micro-kernel

OS Definition, Role, Architecture

- system software placed between hardware and user software (applications)
- roles
  - virtual machine provider (hide hardware and provide abstractions)
  - resource manager
- every resource access / need must be required from OS
  - by calling SO services, i.e. system calls
- OS protects itself from user applications based on different CPU execution modes
  - privileged: kernel mode
  - non-privileged: user mode
- architectures
  - monolith
  - micro-kernel

OS Definition, Role, Architecture

- system software placed between hardware and user software (applications)
- roles
  - virtual machine provider (hide hardware and provide abstractions)
  - resource manager
- every resource access / need must be required from OS
  - by calling SO services, i.e. system calls
- OS protects itself from user applications based on different CPU execution modes
  - privileged: kernel mode
  - non-privileged: user mode
- architectures
  - monolith
  - micro-kernel

OS Definition, Role, Architecture

- system software placed between hardware and user software (applications)
- roles
  - virtual machine provider (hide hardware and provide abstractions)
  - resource manager
- every resource access / need must be required from OS
  - by calling SO services, i.e. system calls
- OS protects itself from user applications based on different CPU execution modes
  - privileged: kernel mode
  - non-privileged: user mode
- architectures
  - monolith
  - micro-kernel

OS Definition, Role, Architecture

- system software placed between hardware and user software (applications)
- roles
  - virtual machine provider (hide hardware and provide abstractions)
  - resource manager
- every resource access / need must be required from OS
  - by calling SO services, i.e. system calls
- OS protects itself from user applications based on different CPU execution modes
  - privileged: kernel mode
  - non-privileged: user mode
- architectures
  - monolith
  - micro-kernel

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

OS Definition, Role, Architecture

- system software placed between hardware and user software (applications)
- roles
  - virtual machine provider (hide hardware and provide abstractions)
  - resource manager
- every resource access / need must be required from OS
  - by calling SO services, i.e. system calls
- OS protects itself from user applications based on different CPU execution modes
  - privileged: kernel mode
  - non-privileged: user mode
- architectures
  - monolith
  - micro-kernel

OS Definition, Role, Architecture

- system software placed between hardware and user software (applications)
- roles
  - virtual machine provider (hide hardware and provide abstractions)
  - resource manager
- every resource access / need must be required from OS
  - by calling SO services, i.e. system calls
- OS protects itself from user applications based on different CPU execution modes
  - privileged: kernel mode
  - non-privileged: user mode
- architectures
  - monolith
  - micro-kernel

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Practice: OS Definition Related Questions

- When is the OS code executed on an uni-processor system? Give examples of at least two situations.
- How is the system call mechanism implemented?
- Which software is run in kernel and user mode respectively in the following two cases?
  - monolithic OS
  - micro-kernel OS

Practice: OS Definition Related Questions

- When is the OS code executed on an uni-processor system? Give examples of at least two situations.
- How is the system call mechanism implemented?
- Which software is run in kernel and user mode respectively in the following two cases?
  - monolithic OS
  - micro-kernel OS

Practice: OS Definition Related Questions

- When is the OS code executed on an uni-processor system? Give examples of at least two situations.
- How is the system call mechanism implemented?
- Which software is run in kernel and user mode respectively in the following two cases?
  - monolithic OS
  - micro-kernel OS

Practice: OS Definition Related Questions

- When is the OS code executed on an uni-processor system? Give examples of at least two situations.
- How is the system call mechanism implemented?
- Which software is run in kernel and user mode respectively in the following two cases?
  - monolithic OS
  - micro-kernel OS

Practice: OS Definition Related Questions

- When is the OS code executed on an uni-processor system? Give examples of at least two situations.
- How is the system call mechanism implemented?
- Which software is run in kernel and user mode respectively in the following two cases?
  - monolithic OS
  - micro-kernel OS

OS Shell

- provides the user interface to the OS
- usually a simple application
- takes user requests and translated them in system calls
- two types
  - graphical
  - text, named command interpreter
- command interpreter functionality
  - get user's command line
  - split it in tokens, i.e. command line and arguments
  - create a new process to execute the specified command
  - waits for created child process' termination
- command line
  - a string of characters separated by spaces
  - first item: command name, actually an executable path
    - searched in directories from PATH
    - security: trust user-established environment (e.g. PATH)
  - other items: command line arguments
  - special characters, like STDIN/OUT redirection, pipe etc.

OS Shell

- provides the user interface to the OS
- usually a simple application
- takes user requests and translated them in system calls
- two types
  - graphical
  - text, named command interpreter
- command interpreter functionality
  - get user's command line
  - split it in tokens, i.e. command line and arguments
  - create a new process to execute the specified command
  - waits for created child process' termination
- command line
  - a string of characters separated by spaces
  - first item: command name, actually an executable path
    - searched in directories from PATH
    - security: trust user-established environment (e.g. PATH)
  - other items: command line arguments
  - special characters, like STDIN/OUT redirection, pipe etc.

OS Shell

- provides the user interface to the OS
- usually a simple application
- takes user requests and translated them in system calls
- two types
  - graphical
  - text, named command interpreter
- command interpreter functionality
  - get user's command line
  - split it in tokens, i.e. command line and arguments
  - create a new process to execute the specified command
  - waits for created child process' termination
- command line
  - a string of characters separated by spaces
  - first item: command name, actually an executable path
  - other items: command line arguments
  - special characters, like STDIN/OUT redirection, pipe etc.

OS Shell

- provides the user interface to the OS
- usually a simple application
- takes user requests and translated them in system calls
- two types
  - graphical
  - text, named command interpreter
- command interpreter functionality
  - get user's command line
  - split it in tokens, i.e. command line and arguments
  - create a new process to execute the specified command
  - waits for created child process' termination
- command line
  - a string of characters separated by spaces
  - first item: command name, actually an executable path
    - searched in directories from PATH
    - security: trust user-established environment (e.g. PATH)
  - other items: command line arguments
  - special characters, like STDIN/OUT redirection, pipe etc.

OS Shell

- provides the user interface to the OS
- usually a simple application
- takes user requests and translated them in system calls
- two types
  - graphical
  - text, named command interpreter
- command interpreter functionality
  - get user's command line
  - split it in tokens, i.e. command line and arguments
  - create a new process to execute the specified command
  - waits for created child process' termination
- command line
  - a string of characters separated by spaces
  - first item: command name, actually an executable path
    - searched in directories from PATH
    - security: trust user-established environment (e.g. PATH)
  - other items: command line arguments
  - special characters, like STDIN/OUT redirection, pipe etc.

OS Shell

- provides the user interface to the OS
- usually a simple application
- takes user requests and translated them in system calls
- two types
  - graphical
  - text, named command interpreter
- command interpreter functionality
  - get user's command line
  - split it in tokens, i.e. command line and arguments
  - create a new process to execute the specified command
  - waits for created child process' termination
- command line
  - a string of characters separated by spaces
  - first item: command name, actually an executable path
  - other items: command line arguments
  - special characters, like STDIN/OUT redirection, pipe etc.

OS Shell

- provides the user interface to the OS
- usually a simple application
- takes user requests and translated them in system calls
- two types
  - graphical
  - text, named command interpreter
- command interpreter functionality
  - get user's command line
  - split it in tokens, i.e. command line and arguments
  - create a new process to execute the specified command
  - waits for created child process' termination
- command line
  - a string of characters separated by spaces
  - first item: command name, actually an executable path
    - searched in directories from PATH
    - security: trust user-established environment (e.g. PATH)
  - other items: command line arguments
  - special characters, like STDIN/OUT redirection, pipe etc.

OS Shell

- provides the user interface to the OS
- usually a simple application
- takes user requests and translated them in system calls
- two types
  - graphical
  - text, named command interpreter
- command interpreter functionality
  - get user's command line
  - split it in tokens, i.e. command line and arguments
  - create a new process to execute the specified command
  - waits for created child process' termination
- command line
  - a string of characters separated by spaces
  - first item: command name, actually an executable path
    - searched in directories from PATH
    - security: trust user-established environment (e.g. PATH)
  - other items: command line arguments
  - special characters, like STDIN/OUT redirection, pipe etc.

OS Shell

- provides the user interface to the OS
- usually a simple application
- takes user requests and translated them in system calls
- two types
  - graphical
  - text, named command interpreter
- command interpreter functionality
  - get user's command line
  - split it in tokens, i.e. command line and arguments
  - create a new process to execute the specified command
  - waits for created child process' termination
- command line
  - a string of characters separated by spaces
  - first item: command name, actually an executable path
  - other items: command line arguments
  - special characters, like STDIN/OUT redirection, pipe etc.

OS Shell

- provides the user interface to the OS
- usually a simple application
- takes user requests and translated them in system calls
- two types
  - graphical
  - text, named command interpreter
- command interpreter functionality
  - get user's command line
  - split it in tokens, i.e. command line and arguments
  - create a new process to execute the specified command
  - waits for created child process' termination
- command line
  - a string of characters separated by spaces
  - first item: command name, actually an executable path
    - searched in directories from PATH
    - security: trust user-established environment (e.g. PATH)
  - other items: command line arguments
  - special characters, like STDIN/OUT redirection, pipe etc.

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

OS Shell

- provides the user interface to the OS
- usually a simple application
- takes user requests and translated them in system calls
- two types
    - graphical
    - text, named command interpreter
- command interpreter functionality
    - get user's command line
    - split it in tokens, i.e. command line and arguments
    - create a new process to execute the specified command
    - waits for created child process' termination
- command line
    - a string of characters separated by spaces
    - first item: command name, actually an executable path
    - other items: command line arguments
    - special characters, like STDIN/OUT redirection, pipe etc.

OS Shell

- provides the user interface to the OS
- usually a simple application
- takes user requests and translated them in system calls
- two types
  - graphical
  - text, named command interpreter
- command interpreter functionality
  - get user's command line
  - split it in tokens, i.e. command line and arguments
  - create a new process to execute the specified command
  - waits for created child process' termination
- command line
  - a string of characters separated by spaces
  - first item: command name, actually an executable path
    - searched in directories from PATH
    - security: trust user-established environment (e.g. PATH)
  - other items: command line arguments
  - special characters, like STDIN/OUT redirection, pipe etc.

OS Shell

- provides the user interface to the OS
- usually a simple application
- takes user requests and translated them in system calls
- two types
  - graphical
  - text, named command interpreter
- command interpreter functionality
  - get user's command line
  - split it in tokens, i.e. command line and arguments
  - create a new process to execute the specified command
  - waits for created child process' termination
- command line
  - a string of characters separated by spaces
  - first item: command name, actually an executable path
    - searched in directories from PATH
    - security: trust user-established environment (e.g. PATH)
  - other items: command line arguments
  - special characters, like STDIN/OUT redirection, pipe etc.

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

OS Shell

- provides the user interface to the OS
- usually a simple application
- takes user requests and translated them in system calls
- two types
  - graphical
  - text, named command interpreter
- command interpreter functionality
  - get user's command line
  - split it in tokens, i.e. command line and arguments
  - create a new process to execute the specified command
  - waits for created child process' termination
- command line
  - a string of characters separated by spaces
  - first item: command name, actually an executable path
    - searched in directories from PATH
    - security: trust user-established environment (e.g. PATH)
  - other items: command line arguments
  - special characters, like STDIN/OUT redirection, pipe etc.

OS Shell

- provides the user interface to the OS
- usually a simple application
- takes user requests and translated them in system calls
- two types
    - graphical
    - text, named command interpreter
- command interpreter functionality
    - get user's command line
    - split it in tokens, i.e. command line and arguments
    - create a new process to execute the specified command
    - waits for created child process' termination
- command line
    - a string of characters separated by spaces
    - first item: command name, actually an executable path
        - searched in directories from PATH
        - security: trust user-established environment (e.g. PATH)
    - other items: command line arguments
    - special characters, like STDIN/OUT redirection, pipe etc.

OS Shell

- provides the user interface to the OS
- usually a simple application
- takes user requests and translated them in system calls
- two types
    - graphical
    - text, named command interpreter
- command interpreter functionality
    - get user's command line
    - split it in tokens, i.e. command line and arguments
    - create a new process to execute the specified command
    - waits for created child process' termination
- command line
    - a string of characters separated by spaces
    - first item: command name, actually an executable path
        - searched in directories from PATH
        - security: trust user-established environment (e.g. PATH)
    - other items: command line arguments
    - special characters, like STDIN/OUT redirection, pipe etc.

OS Shell

- provides the user interface to the OS
- usually a simple application
- takes user requests and translated them in system calls
- two types
  - graphical
  - text, named command interpreter
- command interpreter functionality
  - get user's command line
  - split it in tokens, i.e. command line and arguments
  - create a new process to execute the specified command
  - waits for created child process' termination
- command line
  - a string of characters separated by spaces
  - first item: command name, actually an executable path
    - searched in directories from PATH
    - security: trust user-established environment (e.g. PATH)
  - other items: command line arguments
  - special characters, like STDIN/OUT redirection, pipe etc.

OS Shell

- provides the user interface to the OS
- usually a simple application
- takes user requests and translated them in system calls
- two types
  - graphical
  - text, named command interpreter
- command interpreter functionality
  - get user's command line
  - split it in tokens, i.e. command line and arguments
  - create a new process to execute the specified command
  - waits for created child process' termination
- command line
  - a string of characters separated by spaces
  - first item: command name, actually an executable path
    - searched in directories from PATH
    - security: trust user-established environment (e.g. PATH)
  - other items: command line arguments
  - special characters, like STDIN/OUT redirection, pipe etc.

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Practice: Shell Related Questions

Which is the effect of the following commands?

- ls > file
- read n < file
- ls -R / 1>good 2>err
- cat dict.txt | sort

Practice: Shell Related Questions

Which is the effect of the following commands?

- ls > file
- read n < file
- ls -R / 1>good 2>err
- cat dict.txt | sort

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Practice: Shell Related Questions

Which is the effect of the following commands?

- ls > file
- read n < file
- ls -R / 1>good 2>err
- cat dict.txt | sort

Practice: Shell Related Questions

Which is the effect of the following commands?

- ls > file
- read n < file
- ls -R / 1>good 2>err
- cat dict.txt | sort

File System (FS)

- file concept
  - basic unit of data allocation
  - unstructured stream of bytes
  - file contents managed by user applications, not by OS
  - components: data and meta-data
  - security: too much permissions vulnerability
- directory concept
  - used for organizing the file system space
  - impose file system hierarchy
  - paths: absolute and relative
  - security: path traversal vulnerability
- FS system calls
  - file: open, read, write, lseek, close
  - directory: opendir, readdir, stat, unlink, link
- allocation aspects
  - contiguous allocation ⇒ external fragmentation
  - any-free-block allocation ⇒ internal fragmentation
  - i-nodes, directory entries, links, files with holes

File System (FS)

- file concept
  - basic unit of data allocation
  - unstructured stream of bytes
  - file contents managed by user applications, not by OS
  - components: data and meta-data
  - security: too much permissions vulnerability
- directory concept
  - used for organizing the file system space
  - impose file system hierarchy
  - paths: absolute and relative
  - security: path traversal vulnerability
- FS system calls
  - file: open, read, write, lseek, close
  - directory: opendir, readdir, stat, unlink, link
- allocation aspects
  - contiguous allocation ⇒ external fragmentation
  - any-free-block allocation ⇒ internal fragmentation
  - i-nodes, directory entries, links, files with holes

File System (FS)

- file concept
  - basic unit of data allocation
  - unstructured stream of bytes
  - file contents managed by user applications, not by OS
  - components: data and meta-data
  - security: too much permissions vulnerability
- directory concept
  - used for organizing the file system space
  - impose file system hierarchy
  - paths: absolute and relative
  - security: path traversal vulnerability
- FS system calls
  - file: open, read, write, lseek, close
  - directory: opendir, readdir, stat, unlink, link
- allocation aspects
  - contiguous allocation ⇒ external fragmentation
  - any-free-block allocation ⇒ internal fragmentation
  - i-nodes, directory entries, links, files with holes

File System (FS)

- file concept
  - basic unit of data allocation
  - unstructured stream of bytes
  - file contents managed by user applications, not by OS
  - components: data and meta-data
  - security: too much permissions vulnerability
- directory concept
  - used for organizing the file system space
  - impose file system hierarchy
  - paths: absolute and relative
  - security: path traversal vulnerability
- FS system calls
  - file: open, read, write, lseek, close
  - directory: opendir, readdir, stat, unlink, link
- allocation aspects
  - contiguous allocation ⇒ external fragmentation
  - any-free-block allocation ⇒ internal fragmentation
  - i-nodes, directory entries, links, files with holes

File System (FS)

- file concept
  - basic unit of data allocation
  - unstructured stream of bytes
  - file contents managed by user applications, not by OS
  - components: data and meta-data
  - security: too much permissions vulnerability
- directory concept
  - used for organizing the file system space
  - impose file system hierarchy
  - paths: absolute and relative
  - security: path traversal vulnerability
- FS system calls
  - file: open, read, write, lseek, close
  - directory: opendir, readdir, stat, unlink, link
- allocation aspects
  - contiguous allocation ⇒ external fragmentation
  - any-free-block allocation ⇒ internal fragmentation
  - i-nodes, directory entries, links, files with holes

File System (FS)

- file concept
  - basic unit of data allocation
  - unstructured stream of bytes
  - file contents managed by user applications, not by OS
  - components: data and meta-data
  - security: too much permissions vulnerability
- directory concept
  - used for organizing the file system space
  - impose file system hierarchy
  - paths: absolute and relative
  - security: path traversal vulnerability
- FS system calls
  - file: open, read, write, lseek, close
  - directory: opendir, readdir, stat, unlink, link
- allocation aspects
  - contiguous allocation ⇒ external fragmentation
  - any-free-block allocation ⇒ internal fragmentation
  - i-nodes, directory entries, links, files with holes

File System (FS)

- file concept
  - basic unit of data allocation
  - unstructured stream of bytes
  - file contents managed by user applications, not by OS
  - components: data and meta-data
  - security: too much permissions vulnerability
- directory concept
  - used for organizing the file system space
  - impose file system hierarchy
  - paths: absolute and relative
  - security: path traversal vulnerability
- FS system calls
  - file: open, read, write, lseek, close
  - directory: opendir, readdir, stat, unlink, link
- allocation aspects
  - contiguous allocation ⇒ external fragmentation
  - any-free-block allocation ⇒ internal fragmentation
  - i-nodes, directory entries, links, files with holes

File System (FS)

- file concept
  - basic unit of data allocation
  - unstructured stream of bytes
  - file contents managed by user applications, not by OS
  - components: data and meta-data
  - security: too much permissions vulnerability
- directory concept
  - used for organizing the file system space
  - impose file system hierarchy
  - paths: absolute and relative
  - security: path traversal vulnerability
- FS system calls
  - file: open, read, write, lseek, close
  - directory: opendir, readdir, stat, unlink, link
- allocation aspects
  - contiguous allocation $\Rightarrow$ external fragmentation
  - any-free-block allocation $\Rightarrow$ internal fragmentation
  - i-nodes, directory entries, links, files with holes

File System (FS)

- file concept
  - basic unit of data allocation
  - unstructured stream of bytes
  - file contents managed by user applications, not by OS
  - components: data and meta-data
  - security: too much permissions vulnerability
- directory concept
  - used for organizing the file system space
  - impose file system hierarchy
  - paths: absolute and relative
  - security: path traversal vulnerability
- FS system calls
  - file: open, read, write, lseek, close
  - directory: opendir, readdir, stat, unlink, link
- allocation aspects
  - contiguous allocation ⇒ external fragmentation
  - any-free-block allocation ⇒ internal fragmentation
  - i-nodes, directory entries, links, files with holes

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

File System (FS)

- file concept
  - basic unit of data allocation
  - unstructured stream of bytes
  - file contents managed by user applications, not by OS
  - components: data and meta-data
  - security: too much permissions vulnerability
- directory concept
  - used for organizing the file system space
  - impose file system hierarchy
  - paths: absolute and relative
  - security: path traversal vulnerability
- FS system calls
  - file: open, read, write, lseek, close
  - directory: opendir, readdir, stat, unlink, link
- allocation aspects
  - contiguous allocation ⇒ external fragmentation
  - any-free-block allocation ⇒ internal fragmentation
  - i-nodes, directory entries, links, files with holes

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

File System (FS)

- file concept
  - basic unit of data allocation
  - unstructured stream of bytes
  - file contents managed by user applications, not by OS
  - components: data and meta-data
  - security: too much permissions vulnerability
- directory concept
  - used for organizing the file system space
  - impose file system hierarchy
  - paths: absolute and relative
  - security: path traversal vulnerability
- FS system calls
  - file: open, read, write, lseek, close
  - directory: opendir, readdir, stat, unlink, link
- allocation aspects
  - contiguous allocation ⇒ external fragmentation
  - any-free-block allocation ⇒ internal fragmentation
  - i-nodes, directory entries, links, files with holes

File System (FS)

- file concept
  - basic unit of data allocation
  - unstructured stream of bytes
  - file contents managed by user applications, not by OS
  - components: data and meta-data
  - security: too much permissions vulnerability
- directory concept
  - used for organizing the file system space
  - impose file system hierarchy
  - paths: absolute and relative
  - security: path traversal vulnerability
- FS system calls
  - file: open, read, write, lseek, close
  - directory: opendir, readdir, stat, unlink, link
- allocation aspects
  - contiguous allocation ⇒ external fragmentation
  - any-free-block allocation ⇒ internal fragmentation
  - i-nodes, directory entries, links, files with holes

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

File System (FS)

- file concept
  - basic unit of data allocation
  - unstructured stream of bytes
  - file contents managed by user applications, not by OS
  - components: data and meta-data
  - security: too much permissions vulnerability
- directory concept
  - used for organizing the file system space
  - impose file system hierarchy
  - paths: absolute and relative
  - security: path traversal vulnerability
- FS system calls
  - file: open, read, write, lseek, close
  - directory: opendir, readdir, stat, unlink, link
- allocation aspects
  - contiguous allocation ⇒ external fragmentation
  - any-free-block allocation ⇒ internal fragmentation
  - i-nodes, directory entries, links, files with holes

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

File System (FS)

- file concept
  - basic unit of data allocation
  - unstructured stream of bytes
  - file contents managed by user applications, not by OS
  - components: data and meta-data
  - security: too much permissions vulnerability
- directory concept
  - used for organizing the file system space
  - impose file system hierarchy
  - paths: absolute and relative
  - security: path traversal vulnerability
- FS system calls
  - file: open, read, write, lseek, close
  - directory: opendir, readdir, stat, unlink, link
- allocation aspects
  - contiguous allocation ⇒ external fragmentation
  - any-free-block allocation ⇒ internal fragmentation
  - i-nodes, directory entries, links, files with holes

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

File System (FS)

- file concept
  - basic unit of data allocation
  - unstructured stream of bytes
  - file contents managed by user applications, not by OS
  - components: data and meta-data
  - security: too much permissions vulnerability
- directory concept
  - used for organizing the file system space
  - impose file system hierarchy
  - paths: absolute and relative
  - security: path traversal vulnerability
- FS system calls
  - file: open, read, write, lseek, close
  - directory: opendir, readdir, stat, unlink, link
- allocation aspects
  - contiguous allocation ⇒ external fragmentation
  - any-free-block allocation ⇒ internal fragmentation
  - i-nodes, directory entries, links, files with holes

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

File System (FS)

- file concept
  - basic unit of data allocation
  - unstructured stream of bytes
  - file contents managed by user applications, not by OS
  - components: data and meta-data
  - security: too much permissions vulnerability
- directory concept
  - used for organizing the file system space
  - impose file system hierarchy
  - paths: absolute and relative
  - security: path traversal vulnerability
- FS system calls
  - file: open, read, write, lseek, close
  - directory: opendir, readdir, stat, unlink, link
- allocation aspects
  - contiguous allocation ⇒ external fragmentation
  - any-free-block allocation ⇒ internal fragmentation
  - i-nodes, directory entries, links, files with holes

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

File System (FS)

- file concept
  - basic unit of data allocation
  - unstructured stream of bytes
  - file contents managed by user applications, not by OS
  - components: data and meta-data
  - security: too much permissions vulnerability
- directory concept
  - used for organizing the file system space
  - impose file system hierarchy
  - paths: absolute and relative
  - security: path traversal vulnerability
- FS system calls
  - file: open, read, write, lseek, close
  - directory: opendir, readdir, stat, unlink, link
- allocation aspects
  - contiguous allocation ⇒ external fragmentation
  - any-free-block allocation ⇒ internal fragmentation
  - i-nodes, directory entries, links, files with holes

File System (FS)

- file concept
  - basic unit of data allocation
  - unstructured stream of bytes
  - file contents managed by user applications, not by OS
  - components: data and meta-data
  - security: too much permissions vulnerability
- directory concept
  - used for organizing the file system space
  - impose file system hierarchy
  - paths: absolute and relative
  - security: path traversal vulnerability
- FS system calls
  - file: open, read, write, lseek, close
  - directory: opendir, readdir, stat, unlink, link
- allocation aspects
  - contiguous allocation ⇒ external fragmentation
  - any-free-block allocation ⇒ internal fragmentation
  - i-nodes, directory entries, links, files with holes

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Practice: FS Related Questions

- How is usually the file provided like to the user applications?
- Which of the following extensions usually correspond to text and binary files respectively: html, pdf, c, zip?
- What is an i-node in Linux?
- What does 0640 means in terms of permission rights in Linux?
- Which is the most probable file descriptor returned (and displayed) by the following Linux program? Explain your answer.

```
main()
{
    int fd = open ("/etc/passwd", O_RDONLY);
    printf("fd = %d\n", fd);
}
```

- Write in one line the C code to read a text line from a file, whose file descriptor is given.
- Write the C code to read an integer from offset 16 from a file, whose file descriptor is given.

Process and Thread Management

- process
  - models execution: abstractizes the machine (CPU, memory)
  - describe execution and resources needed for that execution
  - isolates (separates) resources / executions
  - states: running, ready, blocked, terminated
- thread
  - models execution in a process
  - more threads = more concurrent executions in the same process
  - threads of a process share all resources of that process
  - threads useful and effective when
    - logical parallelism exist in the application
    - enough hardware resources available
- scheduling
  - decides who runs and for how long
  - preemptive vs. non-preemptive
    - preemptiveness based on timer interrupt
  - first-come first-served (FCFS), shortest job first (SJB), round-robin (RR), priority-based

Process and Thread Management

- process
  - models execution: abstractizes the machine (CPU, memory)
  - describe execution and resources needed for that execution
  - isolates (separates) resources / executions
  - states: running, ready, blocked, terminated
- thread
  - models execution in a process
  - more threads = more concurrent executions in the same process
  - threads of a process share all resources of that process
  - threads useful and effective when
    - logical parallelism exists in the application
    - enough hardware resources available
- scheduling
  - decides who runs and for how long
  - preemptive vs. non-preemptive
  - first-come first-served (FCFS), shortest job first (SJB), round (RR), priority-based

Process and Thread Management

- process
  - models execution: abstractizes the machine (CPU, memory)
  - describe execution and resources needed for that execution
  - isolates (separates) resources / executions
  - states: running, ready, blocked, terminated
- thread
  - models execution in a process
  - more threads = more concurrent executions in the same process
  - threads of a process share all resources of that process
  - threads useful and effective when
    - logical parallelism exist in the application
    - enough hardware resources available
- scheduling
  - decides who runs and for how long
  - preemptive vs. non-preemptive
    - preemptiveness based on timer interrupt
  - first-come first-served (FCFS), shortest job first (SJB), round-robin (RR), priority-based

Process and Thread Management

- process
  - models execution: abstractizes the machine (CPU, memory)
  - describe execution and resources needed for that execution
  - isolates (separates) resources / executions
  - states: running, ready, blocked, terminated
- thread
  - models execution in a process
  - more threads = more concurrent executions in the same process
  - threads of a process share all resources of that process
  - threads useful and effective when
    - logical parallelism exist in the application
    - enough hardware resources available
- scheduling
  - decides who runs and for how long
  - preemptive vs. non-preemptive
    - preemptiveness based on timer interrupt
  - first-come first-served (FCFS), shortest job first (SJB), round robin (RR), priority-based

Process and Thread Management

- process
  - models execution: abstractizes the machine (CPU, memory)
  - describe execution and resources needed for that execution
  - isolates (separates) resources / executions
  - states: running, ready, blocked, terminated
- thread
  - models execution in a process
  - more threads = more concurrent executions in the same process
  - threads of a process share all resources of that process
  - threads useful and effective when
    - logical parallelism exist in the application
    - enough hardware resources available
- scheduling
  - decides who runs and for how long
  - preemptive vs. non-preemptive
    - preemptiveness based on timer interrupt
  - first-come first-served (FCFS), shortest job first (SJB), round-robin (RR), priority-based

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Process and Thread Management

- process
  - models execution: abstractizes the machine (CPU, memory)
  - describe execution and resources needed for that execution
  - isolates (separates) resources / executions
  - states: running, ready, blocked, terminated
- thread
  - models execution in a process
  - more threads = more concurrent executions in the same process
  - threads of a process share all resources of that process
  - threads useful and effective when
    - logical parallelism exist in the application
    - enough hardware resources available
- scheduling
  - decides who runs and for how long
  - preemptive vs. non-preemptive
  - first-come first-served (FCFS), shortest job first (SJB), round-robin (RR), priority-based

Process and Thread Management

- process
  - models execution: abstractizes the machine (CPU, memory)
  - describe execution and resources needed for that execution
  - isolates (separates) resources / executions
  - states: running, ready, blocked, terminated
- thread
  - models execution in a process
  - more threads = more concurrent executions in the same process
  - threads of a process share all resources of that process
  - threads useful and effective when
    - logical parallelism exist in the application
    - enough hardware resources available
- scheduling
  - decides who runs and for how long
  - preemptive vs. non-preemptive
    - preemptiveness based on timer interrupt
  - first-come first-served (FCFS), shortest job first (SJB), round-robin (RR), priority-based

Process and Thread Management

- process
  - models execution: abstractizes the machine (CPU, memory)
  - describe execution and resources needed for that execution
  - isolates (separates) resources / executions
  - states: running, ready, blocked, terminated
- thread
  - models execution in a process
  - more threads = more concurrent executions in the same process
  - threads of a process share all resources of that process
  - threads useful and effective when
    - logical parallelism exist in the application
    - enough hardware resources available
- scheduling
  - decides who runs and for how long
  - preemptive vs. non-preemptive
    - preemptiveness based on timer interrupt
  - first-come first-served (FCFS), shortest job first (SJB), round-robin (RR), priority-based

Process and Thread Management

- process
  - models execution: abstractizes the machine (CPU, memory)
  - describe execution and resources needed for that execution
  - isolates (separates) resources / executions
  - states: running, ready, blocked, terminated
- thread
  - models execution in a process
  - more threads = more concurrent executions in the same process
  - threads of a process share all resources of that process
  - threads useful and effective when
    - logical parallelism exist in the application
    - enough hardware resources available
- scheduling
  - decides who runs and for how long
  - preemptive vs. non-preemptive
    - preemptiveness based on timer interrupt
  - first-come first-served (FCFS), shortest job first (SJB), round-robin (RR), priority-based

Process and Thread Management

- process
  - models execution: abstractizes the machine (CPU, memory)
  - describe execution and resources needed for that execution
  - isolates (separates) resources / executions
  - states: running, ready, blocked, terminated
- thread
  - models execution in a process
  - more threads = more concurrent executions in the same process
  - threads of a process share all resources of that process
  - threads useful and effective when
    - logical parallelism exist in the application
    - enough hardware resources available
- scheduling
  - decides who runs and for how long
  - preemptive vs. non-preemptive
    - preemptiveness based on timer interrupt
  - first-come first-served (FCFS), shortest job first (SJB), round-robin (RR), priority-based

Process and Thread Management

- process
  - models execution: abstractizes the machine (CPU, memory)
  - describe execution and resources needed for that execution
  - isolates (separates) resources / executions
  - states: running, ready, blocked, terminated
- thread
  - models execution in a process
  - more threads = more concurrent executions in the same process
  - threads of a process share all resources of that process
  - threads useful and effective when
    - logical parallelism exist in the application
    - enough hardware resources available
- scheduling
  - decides who runs and for how long
  - preemptive vs. non-preemptive
    - preemptiveness based on timer interrupt
  - first-come first-served (FCFS), shortest job first (SJB), round-robin (RR), priority-based

Process and Thread Management

- process
  - models execution: abstractizes the machine (CPU, memory)
  - describe execution and resources needed for that execution
  - isolates (separates) resources / executions
  - states: running, ready, blocked, terminated
- thread
  - models execution in a process
  - more threads = more concurrent executions in the same process
  - threads of a process share all resources of that process
  - threads useful and effective when
    - logical parallelism exist in the application
    - enough hardware resources available
- scheduling
  - decides who runs and for how long
  - preemptive vs. non-preemptive
  - first-come first-served (FCFS), shortest job first (SJB), round (RR), priority-based

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Process and Thread Management

- process
  - models execution: abstractizes the machine (CPU, memory)
  - describe execution and resources needed for that execution
  - isolates (separates) resources / executions
  - states: running, ready, blocked, terminated
- thread
  - models execution in a process
  - more threads = more concurrent executions in the same process
  - threads of a process share all resources of that process
  - threads useful and effective when
    - logical parallelism exist in the application
    - enough hardware resources available
- scheduling
  - decides who runs and for how long
  - preemptive vs. non-preemptive
    - preemptiveness based on timer interrupt
  - first-come first-served (FCFS), shortest job first (SJB), round-robin (RR), priority-based

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Process and Thread Management

- process
  - models execution: abstractizes the machine (CPU, memory)
  - describe execution and resources needed for that execution
  - isolates (separates) resources / executions
  - states: running, ready, blocked, terminated
- thread
  - models execution in a process
  - more threads = more concurrent executions in the same process
  - threads of a process share all resources of that process
  - threads useful and effective when
    - logical parallelism exist in the application
    - enough hardware resources available
- scheduling
  - decides who runs and for how long
  - preemptive vs. non-preemptive
    - preemptiveness based on timer interrupt
  - first-come first-served (FCFS), shortest job first (SJB), round-robin (RR), priority-based

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Process and Thread Management

- process
  - models execution: abstractizes the machine (CPU, memory)
  - describe execution and resources needed for that execution
  - isolates (separates) resources / executions
  - states: running, ready, blocked, terminated
- thread
  - models execution in a process
  - more threads = more concurrent executions in the same process
  - threads of a process share all resources of that process
  - threads useful and effective when
    - logical parallelism exist in the application
    - enough hardware resources available
- scheduling
  - decides who runs and for how long
  - preemptive vs. non-preemptive
    - preemptiveness based on timer interrupt
  - first-come first-served (FCFS), shortest job first (SJB), round-robin (RR), priority-based

Process and Thread Management

- process
  - models execution: abstractizes the machine (CPU, memory)
  - describe execution and resources needed for that execution
  - isolates (separates) resources / executions
  - states: running, ready, blocked, terminated
- thread
  - models execution in a process
  - more threads = more concurrent executions in the same process
  - threads of a process share all resources of that process
  - threads useful and effective when
    - logical parallelism exist in the application
    - enough hardware resources available
- scheduling
  - decides who runs and for how long
  - preemptive vs. non-preemptive
    - preemptiveness based on timer interrupt
  - first-come first-served (FCFS), shortest job first (SJB), round-robin (RR), priority-based

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Process and Thread Management

- process
  - models execution: abstractizes the machine (CPU, memory)
  - describe execution and resources needed for that execution
  - isolates (separates) resources / executions
  - states: running, ready, blocked, terminated
- thread
  - models execution in a process
  - more threads = more concurrent executions in the same process
  - threads of a process share all resources of that process
  - threads useful and effective when
    - logical parallelism exist in the application
    - enough hardware resources available
- scheduling
  - decides who runs and for how long
  - preemptive vs. non-preemptive
    - preemptiveness based on timer interrupt
  - first-come first-served (FCFS), shortest job first (SJB), round-robin (RR), priority-based

Practice: Process Related Questions

- How many times is each message in the code below displayed on the screen? Explain your answer.

```
int fd[2];
pipe(fd);
printf("Step 1\n");
fork();
fork();
fork();
printf("Step 2\n");
```

- How many readers and writers, respectively, will exist in the system for the created pipe after the execution of the given code, supposing no process is terminated at that moment?
- Which is the optimum number of threads that should be created by an application to get the best performance (i.e. execution time) when running on a uni-process system and copying a file from one disk to another disk, by encrypting the file contents during the copy operation?

Synchronization Mechanisms

- problem: race conditions of concurrent threads sharing resources
- synchronization means imposing access rules
  - leads to waiting (blocking) $\Rightarrow$ reduce parallelism
- synchronization needs OS and hardware support to get atomicity
- synchronization mechanisms
  - lock $\Rightarrow$ mutual exclusion
  - semaphore
    - generalized lock
    - event counter
  - condition variable
    - specialized waiting mechanism in mutual exclusion area
- classical patterns
  - producers-consumers, readers-writers, rendez-vous (barrier)

Synchronization Mechanisms

- problem: race conditions of concurrent threads sharing resources
- synchronization means imposing access rules
    - leads to waiting (blocking) $\Rightarrow$ reduce parallelism
- synchronization needs OS and hardware support to get atomicity
- synchronization mechanisms
    - lock $\Rightarrow$ mutual exclusion
    - semaphore
        - generalized lock
        - event counter
    - condition variable
        - specialized waiting mechanism in mutual exclusion area
- classical patterns
    - producers-consumers, readers-writers, rendez-vous (barrier)

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Synchronization Mechanisms

- problem: race conditions of concurrent threads sharing resources
- synchronization means imposing access rules
  - leads to waiting (blocking) $\Rightarrow$ reduce parallelism
- synchronization needs OS and hardware support to get atomicity
- synchronization mechanisms
  - lock $\Rightarrow$ mutual exclusion
  - semaphore
    - generalized lock
    - event counter
  - condition variable
    - specialized waiting mechanism in mutual exclusion area
- classical patterns
  - producers-consumers, readers-writers, rendez-vous (barrier)

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Synchronization Mechanisms

- problem: race conditions of concurrent threads sharing resources
- synchronization means imposing access rules
  - leads to waiting (blocking) $\Rightarrow$ reduce parallelism
- synchronization needs OS and hardware support to get atomicity
- synchronization mechanisms
  - lock $\Rightarrow$ mutual exclusion
  - semaphore
    - generalized lock
    - event counter
  - condition variable
    - specialized waiting mechanism in mutual exclusion area
- classical patterns
  - producers-consumers, readers-writers, rendez-vous (barrier)

Synchronization Mechanisms

- problem: race conditions of concurrent threads sharing resources
- synchronization means imposing access rules
  - leads to waiting (blocking) $\Rightarrow$ reduce parallelism
- synchronization needs OS and hardware support to get atomicity
- synchronization mechanisms
  - lock $\Rightarrow$ mutual exclusion
  - semaphore
    - generalized lock
    - event counter
  - condition variable
    - specialized waiting mechanism in mutual exclusion area
- classical patterns
  - producers-consumers, readers-writers, rendez-vous (barrier)

Synchronization Mechanisms

- problem: race conditions of concurrent threads sharing resources
- synchronization means imposing access rules
  - leads to waiting (blocking) ⇒ reduce parallelism
- synchronization needs OS and hardware support to get atomicity
- synchronization mechanisms
  - lock ⇒ mutual exclusion
  - semaphore
    - generalized lock
    - event counter
  - condition variable
    - specialized waiting mechanism in mutual exclusion area
- classical patterns
  - producers-consumers, readers-writers, rendez-vous (barrier)

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Synchronization Mechanisms

- problem: race conditions of concurrent threads sharing resources
- synchronization means imposing access rules
    - leads to waiting (blocking) $\Rightarrow$ reduce parallelism
- synchronization needs OS and hardware support to get atomicity
- synchronization mechanisms
    - lock $\Rightarrow$ mutual exclusion
    - semaphore
        - generalized lock
        - event counter
    - condition variable
        - specialized waiting mechanism in mutual exclusion area
- classical patterns
    - producers-consumers, readers-writers, rendez-vous (barrier)

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Synchronization Mechanisms

- problem: race conditions of concurrent threads sharing resources
- synchronization means imposing access rules
  - leads to waiting (blocking) $\Rightarrow$ reduce parallelism
- synchronization needs OS and hardware support to get atomicity
- synchronization mechanisms
  - lock $\Rightarrow$ mutual exclusion
  - semaphore
    - generalized lock
    - event counter
  - condition variable
    - specialized waiting mechanism in mutual exclusion area
- classical patterns
  - producers-consumers, readers-writers, rendez-vous (barrier)

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Synchronization Mechanisms

- problem: race conditions of concurrent threads sharing resources
- synchronization means imposing access rules
  - leads to waiting (blocking) ⇒ reduce parallelism
- synchronization needs OS and hardware support to get atomicity
- synchronization mechanisms
  - lock ⇒ mutual exclusion
  - semaphore
    - generalized lock
    - event counter
  - condition variable
    - specialized waiting mechanism in mutual exclusion area
- classical patterns
  - producers-consumers, readers-writers, rendez-vous (barrier)

Synchronization Mechanisms

- problem: race conditions of concurrent threads sharing resources
- synchronization means imposing access rules
  - leads to waiting (blocking) $\Rightarrow$ reduce parallelism
- synchronization needs OS and hardware support to get atomicity
- synchronization mechanisms
  - lock $\Rightarrow$ mutual exclusion
  - semaphore
    - generalized lock
    - event counter
  - condition variable
    - specialized waiting mechanism in mutual exclusion area
- classical patterns
  - producers-consumers, readers-writers, rendez-vous (barrier)

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Synchronization Mechanisms

- problem: race conditions of concurrent threads sharing resources
- synchronization means imposing access rules
  - leads to waiting (blocking) $\Rightarrow$ reduce parallelism
- synchronization needs OS and hardware support to get atomicity
- synchronization mechanisms
  - lock $\Rightarrow$ mutual exclusion
  - semaphore
    - generalized lock
    - event counter
  - condition variable
    - specialized waiting mechanism in mutual exclusion area
- classical patterns
  - producers-consumers, readers-writers, rendez-vous (barrier)

Synchronization Mechanisms

- problem: race conditions of concurrent threads sharing resources
- synchronization means imposing access rules
  - leads to waiting (blocking) $\Rightarrow$ reduce parallelism
- synchronization needs OS and hardware support to get atomicity
- synchronization mechanisms
  - lock $\Rightarrow$ mutual exclusion
  - semaphore
    - generalized lock
    - event counter
  - condition variable
    - specialized waiting mechanism in mutual exclusion area
- classical patterns
  - producers-consumers, readers-writers, rendez-vous (barrier)

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Synchronization Mechanisms

- problem: race conditions of concurrent threads sharing resources
- synchronization means imposing access rules
    - leads to waiting (blocking) $\Rightarrow$ reduce parallelism
- synchronization needs OS and hardware support to get atomicity
- synchronization mechanisms
    - lock $\Rightarrow$ mutual exclusion
    - semaphore
        - generalized lock
        - event counter
    - condition variable
        - specialized waiting mechanism in mutual exclusion area
- classical patterns
    - producers-consumers, readers-writers, rendez-vous (barrier)

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Practice: Synchronization Related Questions

- Use semaphores to allow just 10 threads run simultaneously a given function's body.
- Rendezvous: synchronize threads executing functions `boy()` and `girl()` respectively, such that to allow them returning from that functions only in pairs of a "boy" and a "girl".
- Synchronize two concurrent threads executing the two functions below respectively, such that to make them display on the screen the message "*Life is wonderful, isn't is?*"

```
thread_1()
{
    printf("Life ");
    printf("wonderful, ");
}
```

```
thread_2()
{
    printf("is ");
    printf("isn't it?");
}
```

Memory Management

- memory addresses: physical vs. virtual
  - address space
  - virtual address space structure: code, data, heap, stack
- memory binding / translation
  - compile-time (very limited), load-time, run-time (most flexible)
  - need translation tables
- contiguous allocation
  - simple, efficient
  - leads to external fragmentation
  - base and limit registers
- segmentation
  - one contiguous area for each segment (area)
  - segment table
- paging
  - allocates memory in fixed-size chunks ⇒ internal fragmentation
  - virtual pages and physical frames
  - page tables and page table entries
  - page sharing, memory mapped files

Memory Management

- memory addresses: physical vs. virtual
  - address space
  - virtual address space structure: code, data, heap, stack
- memory binding / translation
  - compile-time (very limited), load-time, run-time (most flexible)
  - need translation tables
- contiguous allocation
  - simple, efficient
  - leads to external fragmentation
  - base and limit registers
- segmentation
  - one contiguous area for each segment (area)
  - segment table
- paging
  - allocates memory in fixed-size chunks ⇒ internal fragmentation
  - virtual pages and physical frames
  - page tables and page table entries
  - page sharing, memory mapped files

Memory Management

- memory addresses: physical vs. virtual
  - address space
  - virtual address space structure: code, data, heap, stack
- memory binding / translation
  - compile-time (very limited), load-time, run-time (most flexible)
  - need translation tables
- contiguous allocation
  - simple, efficient
  - leads to external fragmentation
  - base and limit registers
- segmentation
  - one contiguous area for each segment (area)
  - segment table
- paging
  - allocates memory in fixed-size chunks ⇒ internal fragmentation
  - virtual pages and physical frames
  - page tables and page table entries
  - page sharing, memory mapped files

Memory Management

- memory addresses: physical vs. virtual
  - address space
  - virtual address space structure: code, data, heap, stack
- memory binding / translation
  - compile-time (very limited), load-time, run-time (most flexible)
  - need translation tables
- contiguous allocation
  - simple, efficient
  - leads to external fragmentation
  - base and limit registers
- segmentation
  - one contiguous area for each segment (area)
  - segment table
- paging
  - allocates memory in fixed-size chunks $\Rightarrow$ internal fragmentation
  - virtual pages and physical frames
  - page tables and page table entries
  - page sharing, memory mapped files

Memory Management

- memory addresses: physical vs. virtual
  - address space
  - virtual address space structure: code, data, heap, stack
- memory binding / translation
  - compile-time (very limited), load-time, run-time (most flexible)
  - need translation tables
- contiguous allocation
  - simple, efficient
  - leads to external fragmentation
  - base and limit registers
- segmentation
  - one contiguous area for each segment (area)
  - segment table
- paging
  - allocates memory in fixed-size chunks ⇒ internal fragmentation
  - virtual pages and physical frames
  - page tables and page table entries
  - page sharing, memory mapped files

Memory Management

- memory addresses: physical vs. virtual
  - address space
  - virtual address space structure: code, data, heap, stack
- memory binding / translation
  - compile-time (very limited), load-time, run-time (most flexible)
  - need translation tables
- contiguous allocation
  - simple, efficient
  - leads to external fragmentation
  - base and limit registers
- segmentation
  - one contiguous area for each segment (area)
  - segment table
- paging
  - allocates memory in fixed-size chunks ⇒ internal fragmentation
  - virtual pages and physical frames
  - page tables and page table entries
  - page sharing, memory mapped files

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Memory Management

- memory addresses: physical vs. virtual
  - address space
  - virtual address space structure: code, data, heap, stack
- memory binding / translation
  - compile-time (very limited), load-time, run-time (most flexible)
  - need translation tables
- contiguous allocation
  - simple, efficient
  - leads to external fragmentation
  - base and limit registers
- segmentation
  - one contiguous area for each segment (area)
  - segment table
- paging
  - allocates memory in fixed-size chunks ⇒ internal fragmentation
  - virtual pages and physical frames
  - page tables and page table entries
  - page sharing, memory mapped files

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Memory Management

- memory addresses: physical vs. virtual
  - address space
  - virtual address space structure: code, data, heap, stack
- memory binding / translation
  - compile-time (very limited), load-time, run-time (most flexible)
  - need translation tables
- contiguous allocation
  - simple, efficient
  - leads to external fragmentation
  - base and limit registers
- segmentation
  - one contiguous area for each segment (area)
  - segment table
- paging
  - allocates memory in fixed-size chunks $\Rightarrow$ internal fragmentation
  - virtual pages and physical frames
  - page tables and page table entries
  - page sharing, memory mapped files

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Memory Management

- memory addresses: physical vs. virtual
  - address space
  - virtual address space structure: code, data, heap, stack
- memory binding / translation
  - compile-time (very limited), load-time, run-time (most flexible)
  - need translation tables
- contiguous allocation
  - simple, efficient
  - leads to external fragmentation
  - base and limit registers
- segmentation
  - one contiguous area for each segment (area)
  - segment table
- paging
  - allocates memory in fixed-size chunks ⇒ internal fragmentation
  - virtual pages and physical frames
  - page tables and page table entries
  - page sharing, memory mapped files

Memory Management

- memory addresses: physical vs. virtual
  - address space
  - virtual address space structure: code, data, heap, stack
- memory binding / translation
  - compile-time (very limited), load-time, run-time (most flexible)
  - need translation tables
- contiguous allocation
  - simple, efficient
  - leads to external fragmentation
  - base and limit registers
- segmentation
  - one contiguous area for each segment (area)
  - segment table
- paging
  - allocates memory in fixed-size chunks ⇒ internal fragmentation
  - virtual pages and physical frames
  - page tables and page table entries
  - page sharing, memory mapped files

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Memory Management

- memory addresses: physical vs. virtual
  - address space
  - virtual address space structure: code, data, heap, stack
- memory binding / translation
  - compile-time (very limited), load-time, run-time (most flexible)
  - need translation tables
- contiguous allocation
  - simple, efficient
  - leads to external fragmentation
  - base and limit registers
- segmentation
  - one contiguous area for each segment (area)
  - segment table
- paging
  - allocates memory in fixed-size chunks ⇒ internal fragmentation
  - virtual pages and physical frames
  - page tables and page table entries
  - page sharing, memory mapped files

Memory Management

- memory addresses: physical vs. virtual
  - address space
  - virtual address space structure: code, data, heap, stack
- memory binding / translation
  - compile-time (very limited), load-time, run-time (most flexible)
  - need translation tables
- contiguous allocation
  - simple, efficient
  - leads to external fragmentation
  - base and limit registers
- segmentation
  - one contiguous area for each segment (area)
  - segment table
- paging
  - allocates memory in fixed-size chunks $\Rightarrow$ internal fragmentation
  - virtual pages and physical frames
  - page tables and page table entries
  - page sharing, memory mapped files

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Memory Management

- memory addresses: physical vs. virtual
  - address space
  - virtual address space structure: code, data, heap, stack
- memory binding / translation
  - compile-time (very limited), load-time, run-time (most flexible)
  - need translation tables
- contiguous allocation
  - simple, efficient
  - leads to external fragmentation
  - base and limit registers
- segmentation
  - one contiguous area for each segment (area)
  - segment table
- paging
  - allocates memory in fixed-size chunks $\Rightarrow$ internal fragmentation
  - virtual pages and physical frames
  - page tables and page table entries
  - page sharing, memory mapped files

Memory Management

- memory addresses: physical vs. virtual
  - address space
  - virtual address space structure: code, data, heap, stack
- memory binding / translation
  - compile-time (very limited), load-time, run-time (most flexible)
  - need translation tables
- contiguous allocation
  - simple, efficient
  - leads to external fragmentation
  - base and limit registers
- segmentation
  - one contiguous area for each segment (area)
  - segment table
- paging
  - allocates memory in fixed-size chunks $\Rightarrow$ internal fragmentation
  - virtual pages and physical frames
  - page tables and page table entries
  - page sharing, memory mapped files

Memory Management

- memory addresses: physical vs. virtual
  - address space
  - virtual address space structure: code, data, heap, stack
- memory binding / translation
  - compile-time (very limited), load-time, run-time (most flexible)
  - need translation tables
- contiguous allocation
  - simple, efficient
  - leads to external fragmentation
  - base and limit registers
- segmentation
  - one contiguous area for each segment (area)
  - segment table
- paging
  - allocates memory in fixed-size chunks $\Rightarrow$ internal fragmentation
  - virtual pages and physical frames
  - page tables and page table entries
  - page sharing, memory mapped files

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Memory Management

- memory addresses: physical vs. virtual
  - address space
  - virtual address space structure: code, data, heap, stack
- memory binding / translation
  - compile-time (very limited), load-time, run-time (most flexible)
  - need translation tables
- contiguous allocation
  - simple, efficient
  - leads to external fragmentation
  - base and limit registers
- segmentation
  - one contiguous area for each segment (area)
  - segment table
- paging
  - allocates memory in fixed-size chunks ⇒ internal fragmentation
  - virtual pages and physical frames
  - page tables and page table entries
  - page sharing, memory mapped files

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Memory Management

- memory addresses: physical vs. virtual
  - address space
  - virtual address space structure: code, data, heap, stack
- memory binding / translation
  - compile-time (very limited), load-time, run-time (most flexible)
  - need translation tables
- contiguous allocation
  - simple, efficient
  - leads to external fragmentation
  - base and limit registers
- segmentation
  - one contiguous area for each segment (area)
  - segment table
- paging
  - allocates memory in fixed-size chunks $\Rightarrow$ internal fragmentation
  - virtual pages and physical frames
  - page tables and page table entries
  - page sharing, memory mapped files

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Memory Management

- memory addresses: physical vs. virtual
  - address space
  - virtual address space structure: code, data, heap, stack
- memory binding / translation
  - compile-time (very limited), load-time, run-time (most flexible)
  - need translation tables
- contiguous allocation
  - simple, efficient
  - leads to external fragmentation
  - base and limit registers
- segmentation
  - one contiguous area for each segment (area)
  - segment table
- paging
  - allocates memory in fixed-size chunks $\Rightarrow$ internal fragmentation
  - virtual pages and physical frames
  - page tables and page table entries
  - page sharing, memory mapped files

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Practice: Memory Mng Related Questions

- Which is the size in bytes of a process' virtual address space on a system using 64 bits for a memory address?
- How may page table entries must be used by such a system to map a process VAS, supposing the page size to be 4MB?
- Illustrate on such a system the part of a process' VAS and page table (entries) used for mapping the memory required be the following code:

```
unsigned int *p = malloc(4*4*1024*1024);
printf("p=%u\n", p);      // displays p = 1000*4*1024*1024
```

- Which page does the following instruction refer to? Could it be executed successfully or not?

```
p[4*1024*1024] = 10;
```

Security Aspects

- untrusted application environment
  - untrusted PATH variable
- file system
  - too much permissions
  - path traversal
- memory-related: bad / wrong memory accesses
  - NULL-pointer usage
  - use-after free
  - buffer overflow

Security Aspects

- untrusted application environment
    - untrusted PATH variable
- file system
    - too much permissions
    - path traversal
- memory-related: bad / wrong memory accesses
    - NULL-pointer usage
    - use-after free
    - buffer overflow

Security Aspects

- untrusted application environment
  - untrusted PATH variable
- file system
  - too much permissions
  - path traversal
- memory-related: bad / wrong memory accesses
  - NULL-pointer usage
  - use-after free
  - buffer overflow

Security Aspects

- untrusted application environment
  - untrusted PATH variable
- file system
  - too much permissions
  - path traversal
- memory-related: bad / wrong memory accesses
  - NULL-pointer usage
  - use-after free
  - buffer overflow

Security Aspects

- untrusted application environment
  - untrusted PATH variable
- file system
  - too much permissions
  - path traversal
- memory-related: bad / wrong memory accesses
  - NULL-pointer usage
  - use-after free
  - buffer overflow

Security Aspects

- untrusted application environment
  - untrusted PATH variable
- file system
  - too much permissions
  - path traversal
- memory-related: bad / wrong memory accesses
  - NULL-pointer usage
  - use-after free
  - buffer overflow

Security Aspects

- untrusted application environment
  - untrusted PATH variable
- file system
  - too much permissions
  - path traversal
- memory-related: bad / wrong memory accesses
  - NULL-pointer usage
  - use-after free
  - buffer overflow

Security Aspects

- untrusted application environment
  - untrusted PATH variable
- file system
  - too much permissions
  - path traversal
- memory-related: bad / wrong memory accesses
  - NULL-pointer usage
  - use-after free
  - buffer overflow

Security Aspects

- untrusted application environment
  - untrusted PATH variable
- file system
  - too much permissions
  - path traversal
- memory-related: bad / wrong memory accesses
  - NULL-pointer usage
  - use-after free
  - buffer overflow

Outline

Outline

1. OS Subject Review
2. Conclusions
   - Past
   - Future

What did we talk about?

- OS's place and role (and definition), relative to hardware and other software
- OS structure
- Shell, i.e. command interpreter
- File System
- Process Management
- Memory Management
- Security Aspects

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

What have we learned?

- basic concepts like
  - process, thread — execution, resources, isolation, scheduling
  - file: storage, sequence of bytes (no format), meta-data (i-node), links, fragmentation, open files
  - directory: organization, file tree, collection of directory entries
  - synchronization: locks (mutex), semaphores, condition variables, producers/consumers, readers/writers, barrier
  - memory: address space, virtual/physical memory addresses, ELF, paging
  - IPC mechanisms: pipes, shared memory
  - fragmentation: external (contiguous allocation, best-fit, worst-fit), internal
  - basic security issues: buffer overflow, path traversal
- system calls to access various OS (Linux) services
- write C programs to have access to Linux system calls

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

What can we do?

- explain OS functionality and concepts
- understand better applications functionality, their relationship with the OS and some of their crash reasons
- write (if needed) C programs to access low-level OS services
- choose the appropriate OS for a particular purpose
- tune better an OS for particular applications/context/purposes

What can we do?

- explain OS functionality and concepts
- understand better applications functionality, their relationship with the OS and some of their crash reasons
- write (if needed) C programs to access low-level OS services
- choose the appropriate OS for a particular purpose
- tune better an OS for particular applications/context/purposes

What can we do?

- explain OS functionality and concepts
- understand better applications functionality, their relationship with the OS and some of their crash reasons
- write (if needed) C programs to access low-level OS services
- choose the appropriate OS for a particular purpose
- tune better an OS for particular applications/context/purposes

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

What can we do?

- explain OS functionality and concepts
- understand better applications functionality, their relationship with the OS and some of their crash reasons
- write (if needed) C programs to access low-level OS services
- choose the appropriate OS for a particular purpose
- tune better an OS for particular applications/context/purposes

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

What can we do?

- explain OS functionality and concepts
- understand better applications functionality, their relationship with the OS and some of their crash reasons
- write (if needed) C programs to access low-level OS services
- choose the appropriate OS for a particular purpose
- tune better an OS for particular applications/context/purposes

Outline

Will I need OS knowledge?

- all the time: understand, explain, evaluate, configure

Will I use OS system calls?

- sometimes, especially when you need a particular (efficient, lower-level) functionality
- all the time, if you work at low-level (in C, asm)

Will I use OS system calls?

- sometimes, especially when you need a particular (efficient, lower-level) functionality
- all the time, if you work at low-level (in C, asm)

Operating System Design

- this is (our) next step: the advanced course (UTCN, CS Dept.) regarding OS
- what about: internals of an OS, design/implementation alternatives of
  - scheduling algorithms and synchronization mechanisms
  - user processes and threads
  - system calls for open files, processes and threads
  - memory management, virtual memory, page replacement algorithms
  - file system
- practical aspects: design, implement (in C) and test an OS (*HAL9000* — UTCN; *Pintos* — Stanford, USA); learn how to debug an OS on a remote virtual machine
- practical aspects: work in team (3 members)

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Operating System Design

- this is (our) next step: the advanced course (UTCN, CS Dept.) regarding OS
- what about: internals of an OS, design/implementation alternatives of
  - scheduling algorithms and synchronization mechanisms
  - user processes and threads
  - system calls for open files, processes and threads
  - memory management, virtual memory, page replacement algorithms
  - file system
- practical aspects: design, implement (in C) and test an OS (*HAL9000* — UTCN; *Pintos* — Stanford, USA); learn how to debug an OS on a remote virtual machine
- practical aspects: work in team (3 members)

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Operating System Design

- this is (our) next step: the advanced course (UTCN, CS Dept.) regarding OS
- what about: internals of an OS, design/implementation alternatives of
  - scheduling algorithms and synchronization mechanisms
  - user processes and threads
  - system calls for open files, processes and threads
  - memory management, virtual memory, page replacement algorithms
  - file system
- practical aspects: design, implement (in C) and test an OS (*HAL9000* — UTCN; *Pintos* — Stanford, USA); learn how to debug an OS on a remote virtual machine
- practical aspects: work in team (3 members)

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Operating System Design

- this is (our) next step: the advanced course (UTCN, CS Dept.) regarding OS
- what about: internals of an OS, design/implementation alternatives of
  - scheduling algorithms and synchronization mechanisms
  - user processes and threads
  - system calls for open files, processes and threads
  - memory management, virtual memory, page replacement algorithms
  - file system
- practical aspects: design, implement (in C) and test an OS (*HAL9000* — UTCN; *Pintos* — Stanford, USA); learn how to debug an OS on a remote virtual machine
- practical aspects: work in team (3 members)

Operating System Design

- this is (our) next step: the advanced course (UTCN, CS Dept.) regarding OS
- what about: internals of an OS, design/implementation alternatives of
  - scheduling algorithms and synchronization mechanisms
  - user processes and threads
  - system calls for open files, processes and threads
  - memory management, virtual memory, page replacement algorithms
  - file system
- practical aspects: design, implement (in C) and test an OS (*HAL9000* — UTCN; *Pintos* — Stanford, USA); learn how to debug an OS on a remote virtual machine
- practical aspects: work in team (3 members)

Operating System Design

- this is (our) next step: the advanced course (UTCN, CS Dept.) regarding OS
- what about: internals of an OS, design/implementation alternatives of
  - scheduling algorithms and synchronization mechanisms
  - user processes and threads
  - system calls for open files, processes and threads
  - memory management, virtual memory, page replacement algorithms
  - file system
- practical aspects: design, implement (in C) and test an OS (*HAL9000* — UTCN; *Pintos* — Stanford, USA); learn how to debug an OS on a remote virtual machine
- practical aspects: work in team (3 members)

Operating System Design

- this is (our) next step: the advanced course (UTCN, CS Dept.) regarding OS
- what about: internals of an OS, design/implementation alternatives of
  - scheduling algorithms and synchronization mechanisms
  - user processes and threads
  - system calls for open files, processes and threads
  - memory management, virtual memory, page replacement algorithms
  - file system
- practical aspects: design, implement (in C) and test an OS (*HAL9000* — UTCN; *Pintos* — Stanford, USA); learn how to debug an OS on a remote virtual machine
- practical aspects: work in team (3 members)

Operating System Design

- this is (our) next step: the advanced course (UTCN, CS Dept.) regarding OS
- what about: internals of an OS, design/implementation alternatives of
  - scheduling algorithms and synchronization mechanisms
  - user processes and threads
  - system calls for open files, processes and threads
  - memory management, virtual memory, page replacement algorithms
  - file system
- practical aspects: design, implement (in C) and test an OS (*HAL9000* — UTCN; *Pintos* — Stanford, USA); learn how to debug an OS on a remote virtual machine
- practical aspects: work in team (3 members)

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Operating System Design

- this is (our) next step: the advanced course (UTCN, CS Dept.) regarding OS
- what about: internals of an OS, design/implementation alternatives of
  - scheduling algorithms and synchronization mechanisms
  - user processes and threads
  - system calls for open files, processes and threads
  - memory management, virtual memory, page replacement algorithms
  - file system
- practical aspects: design, implement (in C) and test an OS (*HAL9000* — UTCN; *Pintos* — Stanford, USA); learn how to debug an OS on a remote virtual machine
- practical aspects: work in team (3 members)

Operating System Design (cont.)

- usefulness
  - an OS is a complex management software; all the management algorithms and techniques work for many other complex management software even at higher levels
  - understanding low-level mechanisms makes you understand better higher-level ones' behavior
- it is an 4th year *optional subject*
- *students' myth*: (highly) difficult
- *reality (trust me)*: just interesting and challenging, not more difficult than other subjects/projects

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Operating System Design (cont.)

- usefulness
  - an OS is a complex management software; all the management algorithms and techniques work for many other complex management software even at higher levels
  - understanding low-level mechanisms makes you understand better higher-level ones' behavior
- it is an 4th year *optional subject*
- *students' myth*: (highly) difficult
- *reality (trust me)*: just interesting and challenging, not more difficult than other subjects/projects

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Operating System Design (cont.)

- usefulness
  - an OS is a complex management software; all the management algorithms and techniques work for many other complex management software even at higher levels
  - understanding low-level mechanisms makes you understand better higher-level ones' behavior
- it is an 4th year *optional subject*
- *students' myth*: (highly) difficult
- *reality (trust me)*: just interesting and challenging, not more difficult than other subjects/projects

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Operating System Design (cont.)

- usefulness
  - an OS is a complex management software; all the management algorithms and techniques work for many other complex management software even at higher levels
  - understanding low-level mechanisms makes you understand better higher-level ones' behavior
- it is an 4th year *optional subject*
- *students' myth*: (highly) difficult
- *reality (trust me)*: just interesting and challenging, not more difficult than other subjects/projects

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Operating System Design (cont.)

- usefulness
  - an OS is a complex management software; all the management algorithms and techniques work for many other complex management software even at higher levels
  - understanding low-level mechanisms makes you understand better higher-level ones' behavior
- it is an 4th year *optional subject*
- *students' myth*: (highly) difficult
- *reality (trust me)*: just interesting and challenging, not more difficult than other subjects/projects

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Operating System Design (cont.)

- usefulness
  - an OS is a complex management software; all the management algorithms and techniques work for many other complex management software even at higher levels
  - understanding low-level mechanisms makes you understand better higher-level ones' behavior
- it is an 4th year *optional subject*
- *students' myth*: (highly) difficult
- *reality (trust me)*: just interesting and challenging, not more difficult than other subjects/projects

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Operating System Design (cont.)

- I would like you to think like this:
  - “I’d choose this subject just because I am really interested in, it sounds great and challenging, I feel I need it in my future career, I want to understand how low-level mechanisms and aspects work, it fits my aptitudes and interests, I have heard about learning useful things etc.”
- I would not like to hear about you saying:
  - “I DID NOT choose that subject just because I have a job and do not have enough time for it, I’ve heard it takes long(er) to solve the assignments, I’ve heard it is (more) difficult etc.”

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Operating System Design (cont.)

- I would like you to think like this:
  - "I'd choose this subject just because I am really interested in, it sounds great and challenging, I feel I need it in my future career, I want to understand how low-level mechanisms and aspects work, it fits my aptitudes and interests, I have heard about learning useful things etc."
- I would not like to hear about you saying:
  - "I DID NOT choose that subject just because I have a job and do not have enough time for it, I've heard it takes long(er) to solve the assignments, I've heard it is (more) difficult etc."

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Operating System Design (cont.)

- I would like you to think like this:
  - "I'd choose this subject just because I am really interested in, it sounds great and challenging, I feel I need it in my future career, I want to understand how low-level mechanisms and aspects work, it fits my aptitudes and interests, I have heard about learning useful things etc."
- I would not like to hear about you saying:
  - "I DID NOT choose that subject just because I have a job and do not have enough time for it, I've heard it takes long(er) to solve the assignments, I've heard it is (more) difficult etc."

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Operating System Design (cont.)

- I would like you to think like this:
  - "I'd choose this subject just because I am really interested in, it sounds great and challenging, I feel I need it in my future career, I want to understand how low-level mechanisms and aspects work, it fits my aptitudes and interests, I have heard about learning useful things etc."
- I would not like to hear about you saying:
  - "I DID NOT choose that subject just because I have a job and do not have enough time for it, I've heard it takes long(er) to solve the assignments, I've heard it is (more) difficult etc."

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Security Master Program (SISC)

- many OS-related courses
  - build low-level OS layers on the 64-bit architecture
  - kernel driver development
  - build virtualization security-oriented OS (hypervisor)
- many security-related aspects
  - secure coding
  - Web security
  - mobile systems (Android) security
  - big-data and security
  - penetration testing
  - risk management
  - cryptography

Security Master Program (SISC)

- many OS-related courses
  - build low-level OS layers on the 64-bit architecture
  - kernel driver development
  - build virtualization security-oriented OS (hypervisor)
- many security-related aspects
  - secure coding
  - Web security
  - mobile systems (Android) security
  - big-data and security
  - penetration testing
  - risk management
  - cryptography

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Security Master Program (SISC)

- many OS-related courses
  - build low-level OS layers on the 64-bit architecture
  - kernel driver development
  - build virtualization security-oriented OS (hypervisor)
- many security-related aspects
  - secure coding
  - Web security
  - mobile systems (Android) security
  - big-data and security
  - penetration testing
  - risk management
  - cryptography

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Security Master Program (SISC)

- many OS-related courses
  - build low-level OS layers on the 64-bit architecture
  - kernel driver development
  - build virtualization security-oriented OS (hypervisor)
- many security-related aspects
  - secure coding
  - Web security
  - mobile systems (Android) security
  - big-data and security
  - penetration testing
  - risk management
  - cryptography

Security Master Program (SISC)

- many OS-related courses
  - build low-level OS layers on the 64-bit architecture
  - kernel driver development
  - build virtualization security-oriented OS (hypervisor)
- many security-related aspects
  - secure coding
  - Web security
  - mobile systems (Android) security
  - big-data and security
  - penetration testing
  - risk management
  - cryptography

Security Master Program (SISC)

- many OS-related courses
  - build low-level OS layers on the 64-bit architecture
  - kernel driver development
  - build virtualization security-oriented OS (hypervisor)
- many security-related aspects
  - secure coding
  - Web security
  - mobile systems (Android) security
  - big-data and security
  - penetration testing
  - risk management
  - cryptography

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Security Master Program (SISC)

- many OS-related courses
  - build low-level OS layers on the 64-bit architecture
  - kernel driver development
  - build virtualization security-oriented OS (hypervisor)
- many security-related aspects
  - secure coding
  - Web security
  - mobile systems (Android) security
  - big-data and security
  - penetration testing
  - risk management
  - cryptography

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Security Master Program (SISC)

- many OS-related courses
  - build low-level OS layers on the 64-bit architecture
  - kernel driver development
  - build virtualization security-oriented OS (hypervisor)
- many security-related aspects
  - secure coding
  - Web security
  - mobile systems (Android) security
  - big-data and security
  - penetration testing
  - risk management
  - cryptography

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Security Master Program (SISC)

- many OS-related courses
  - build low-level OS layers on the 64-bit architecture
  - kernel driver development
  - build virtualization security-oriented OS (hypervisor)
- many security-related aspects
  - secure coding
  - Web security
  - mobile systems (Android) security
  - big-data and security
  - penetration testing
  - risk management
  - cryptography

Security Master Program (SISC)

- many OS-related courses
  - build low-level OS layers on the 64-bit architecture
  - kernel driver development
  - build virtualization security-oriented OS (hypervisor)
- many security-related aspects
  - secure coding
  - Web security
  - mobile systems (Android) security
  - big-data and security
  - penetration testing
  - risk management
  - cryptography

Security Master Program (SISC)

- many OS-related courses
  - build low-level OS layers on the 64-bit architecture
  - kernel driver development
  - build virtualization security-oriented OS (hypervisor)
- many security-related aspects
  - secure coding
  - Web security
  - mobile systems (Android) security
  - big-data and security
  - penetration testing
  - risk management
  - cryptography

Security Master Program (SISC)

- many OS-related courses
  - build low-level OS layers on the 64-bit architecture
  - kernel driver development
  - build virtualization security-oriented OS (hypervisor)
- many security-related aspects
  - secure coding
  - Web security
  - mobile systems (Android) security
  - big-data and security
  - penetration testing
  - risk management
  - cryptography

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Finish

That's all folks! So long, folks!