COT 4600 Operating Systems Fall 2009

Dan C. Marinescu

Office: HEC 439 B

Office hours: Tu-Th 3:00-4:00 PM

Lecture 10

- Last time:
  - Modular Sharing
  - Metadata and Name Overloading
  - Addresses Case study: the Unix file system
- Today:
  - User-friendly names; Lifetime of names
  - Case study: URL
  - Soft modularity
  - Hard modularity
- Next Time:
  - Client/Service organization

User-friendly names

- Are always strings of characters.
- Requirements:
  - the need for a name to be unique
  - easy to remember
  - use of lower and upper case letters more common today
  - Multics → first system to use both upper and lower case characters.
- Case sensitive versus case-preserving names
  - http://Jupiter.Athena.MIT.edu is different from http://jupiter.athena.mit.edu
- Icons acting as a name

Relative lifetime of names

- The cardinality of the name-space can be limited (the total number of characters is limited, the alphabet is restricted, etc).
  - Names permanently bound to objects; e.g., the registers of a processor
  - Names have a limited life-time and must be reused; dynamic IP addresses are leased for limited amount of time
- In many cases the cardinality of the name-space is practically unlimited and arbitrary names can be choosen.
- The binding of the name and the object
  - Long-lived → the phone number, the Email address
  - Short-lived → the label on your cup of coffee at Starbucks
  - May be renewed → the binding of the system calls (which are long-lived) issued by an application program to the operating system (alos a long-lived object) are renewed every time the program runs
- Dangling references →names that no longer correspond to existing objects; e.g., old telephone numbers.
- Orphan object → an object that outlives its name. How to garbage collect the space used by the object?

Case study: URLs

- Hyperlink →
  - a reference in a document to an external or internal piece of information
  - makes a logical connection between two places in the same or different documents
  - to browse through web pages some text in the current document is highlighted so that when clicked, the browser automatically displays another page or changes the current page to show the referenced content.
  - the basic building block of **hypertexts**.
- The Web → a file transfer protocol using
  - The Hypertext Markup language (HTML) to describe the contents
  - The Hypertext Transfer Protocol (HTTP) for communication.
- Universal Resource Locator (URL)
  - A name in the URL name space
  - Absolute URL → an URL which carries its own context.
  - Example :

    http://www.cs.ucf.edu/~dcm/Teaching/OperatingSystemsCOT4600/ClassIndex.html
  - Relative URL → an URL which does not carry its own context. The context is derived from the page where this relative URL occurs.
  - Example : <a href =“Projects..htm”? Link to projects.

The name-mapping algorithm for an URL

- Example:
  http://www.cs.ucf.edu/~dcm/Teaching/OperatingSystemsCOT4600/ClassIndex.html
- The string before ":" (colon) → identifies the protocol used, e.g., "http".
- The string between "//" and the following "/" → the host name passed to DNS (Domain Name Server) to resolve (e.g., www.cs.ucf.edu)
- The browser uses the protocol (in our case http) to open the connection with the http-server on the host with the IP address returned by DNS and to locate the file /~dcm/Teaching/OperatingSystemsCOT4600/ClassIndex.html
- If the file is found then the http-server send the file

URL case sensitivity

- The host name part of the URL (the one sent to DNS) is case sensitive.
- The part of the URL used to locate the object on the host (the path name)
  - Depends upon the protocol used
  - HTTP says “this string is NOT a Unix file name” but does not say anything about case sensitivity
  - Case sensitivity for HTTP depends on the file system of the server
    - On a standard Unix system the path name is case sensitive
    - On a MAC the path name is case-preserving

Alternatives to URL

- Permanent URL (PURL) →
- Universal Resource Name (URN) → The URNs are part of a larger Internet information architecture which is composed of
  - URNs used for identification,
  - URCs for including meta-information.
  - URLs for locating or finding resources.
- Digital Object Identifier (DOJ) → includes metadata; diverse granularity

Soft modularity

- Modularity is critical but the techniques discussed so far do not limit common errors.
- Soft modularity → divide a program in procedures that call each other.
  - Hard to debug; if one of the modules has an infinite loop, a call never returns
  - The caller and the callee are in the same address space and may misuse the stack.
  - Naming conflicts and wrong context specification.

Example:

```
procedure MEASURE (func)
    start_time ← GET_TIME(SECONDS)
    funct()
    end_time ←GET_TIME(SECONDS)
return (end_time-start_time)

procedure GET_TIME (units)
    time←CLOCK
    time ← CONVERT_TO_UNITS(time,units)
return time
```

Machine code for MEASURE

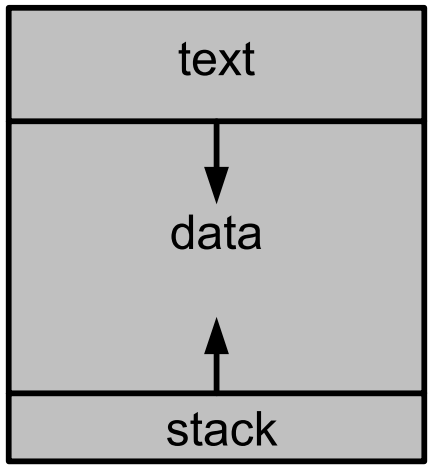
```
100    ST      R1,SP          //save content of R1 on the stack
104    ADD     4, SP          //increment  stack pointer
108    ST      R2, SP         //save content of R2 on the stack
112    ADD     4, SP          //increment  stack pointer
116    LA      R1, SECONDS    //load address of the argument in R1
120    ST      R1, SP         // store address of the argument on the stack
124    ADD     4, SP          // increment stack pointer
128    L A     R1,148         // load return address in R1
132    ST      R1, SP         // store return address on the stack
136    ADD     4, SP          //adjust top stack  pointer
140    L A     R1, 200        // load address of GET_TIME in R1
144    JMP     R1             //transfer control to GET_TIME
148    S       4,SP           // decrement stack pointer
152    L       R2, SP         // restore the contents of R2
156    S       4,SP           // decrement stack pointer
160    L       R1,SP          // restore the contents of R1
164    S       4,SP           // decrement stack pointer
168    ST      R0, start      // store result passed by GET_TIME in Ro into start
```

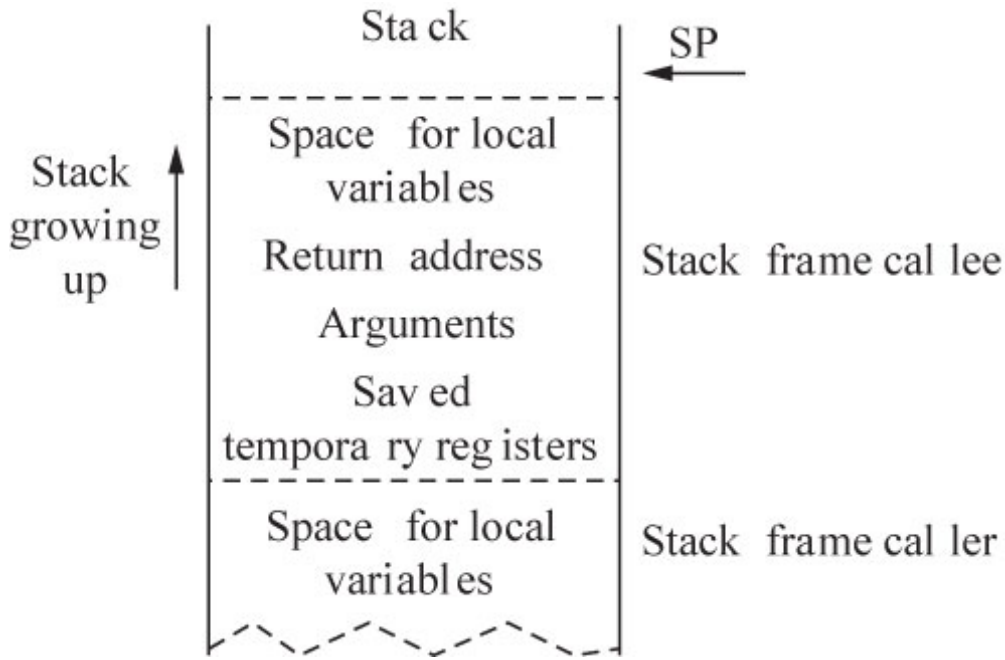
Machine code for GET_TIME

```
200     L       R1,SP           //load address of the stack pointer  in R1
204     S       R1,8            //increment  stack pointer
208     L       R2, R1          //load address of the argument in R2
212     code for the body of GET_TIME
216     code for the body of GET_TIME
220     L       R0, time        // load in R0 the result
224     L       R1,SP           // reload in R1 address of the stack pointer
228     S       R1,4            // decrement the stack pointer
231     L       PC,R1           // load return address from stack into PC
```

Procedure call convention

1. Caller
   1. saves on the stack (after each operation it adjusts the SP)
      1. registers
      2. arguments
      3. return address
   2. transfers control to the calle (jump to its starting address)
2. Calee
   1. loads from the stack the arguments
   2. carries out the desired calculation and load the results in a register (R0)
   3. transfers control back to the caller →loads in the PC the return address to the caller
3. Caller
   1. adjusts the stack
   2. restores its registers


text
data
stack


Sta ck
SP
Space for local variabl es
Stack growing up
Return address
Arguments
Sav ed tempora ry reg isters
Stack frame cal lee
Space for local variabl es
Stack frame cal ler

Soft modularity allows errors to propagate

- Conventions between caller and callee regarding register usage:
  - The caller passes the argument (the address of the variable SECONDS) in register R1
  - The callee returns the value of the result in register R0
  - The callee uses register R2 so the caller must save it before transferring control to the callee
- Potential problems caused by soft modularity
  - The callee is expected to leave the stack pointer as it was set by the caller. But the callee may mess up…
  - The transfer of control→ the callee may return to the wrong address
  - The caller may attempt to get the result from the wrong register
  - The callee may use registers that the caller has not saved on the stack before transferring control to the callee
  - An error of the callee will affect the caller
  - If either the caller or the callee agree to communicate using global variables then changing of these variable will affect other modules

Strongly typed languages help enforce modularity

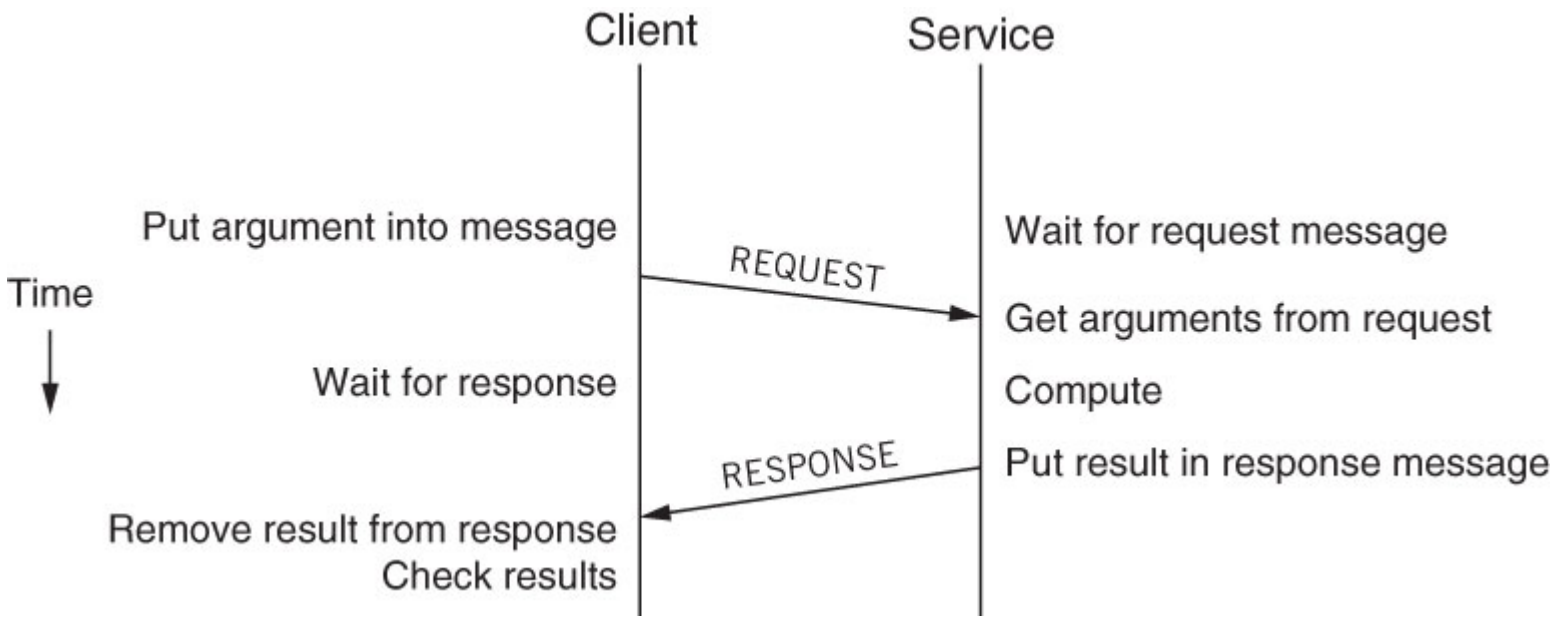
- Provide:
  - Strong guarantees about the run-time behavior of a program before program execution, whether provided by static analysis, the execution semantics of the language or another mechanism.
  - Type safety; that is, at compile or run time, the rejection of operations or function calls which attempt to disregard data types.
  - The guarantee that a well-defined error or exceptional behavior occurs as soon as a type-matching failure happens at runtime.
  - The compiler ensures that operations only occur on operand types that are valid for the operation.
  - The type of a given data object does not vary over that object's lifetime. For example, class instances may not have their class altered.
  - The absence of ways to evade the type system. Such evasions are possible in languages that allow programmer access to the underlying representation of values, i.e., their bit-pattern.
  - A programming language is strongly typed if type conversions are allowed only when an explicit notation, often called a *cast*, is used to indicate the desire of converting one type to another.
  - Disallowing any kind of type conversion. Values of one type cannot be converted to another type, explicitly or implicitly.

Soft modularity may be affected by other factors

- Different modules are written in different languages
- Errors in the run-time support
- Errors in the compiler

Enforced modularity

- Enforced modularity → force modules to interact only by sending messages.
- The client/service organization makes it more difficult:
  - For programmers to violate modularity → the only way two modules interact is by means of messages; naming within one module are not visible outside the module.
  - Errors to propagate → clients and services are independent modules and may fail separately.
  - Attack the system → if messages are checked carefully the attaker has a very hard time
- Other advantages:
  - The system is more robust; the servers are stateless.
  - Resources can be managed more effectively.


Client
Service
Put argument into message
REQUEST
Wait for request message
Time
Get arguments from request
Wait for response
Compute
RESPONSE
Put result in response message
Remove result from response
Check results