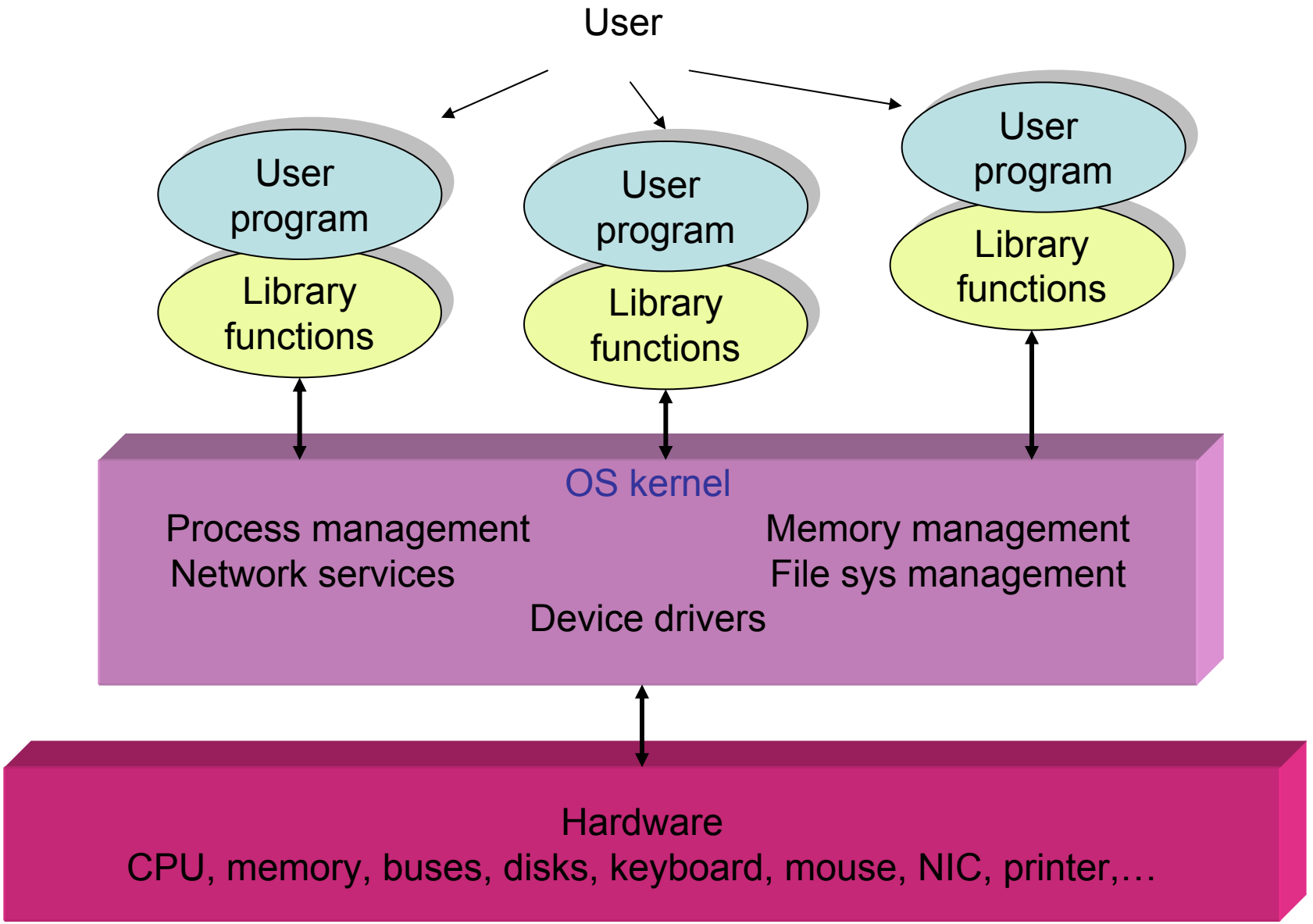
Operating systems II
CS 2506

Dr. Dan Grigoras
Computer Science Department

UNIVERSITY COLLEGE, CORK
Coláiste na hOllscoile Corcaigh

System architecture

- What is the relationship between the applications/programs and “the computing system” ?
- How can a user program access services of the operating system ?
- What is a process ?
- What is a thread and why it is useful ?


User
User program
Library functions
User program
Library functions
User program
Library functions
OS kernel
Process management
Network services
Memory management
File sys management
Device drivers
Hardware
CPU, memory, buses, disks, keyboard, mouse, NIC, printer,…

Layers

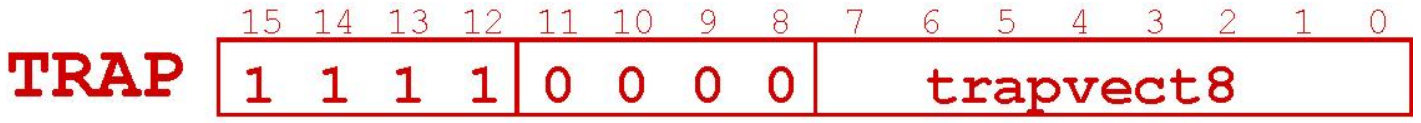
- The bottom layer is the hardware; it accepts primitive commands such as “seek the disk arm to track 79, select head 3 and read sector 5”. Software that interacts directly with the hardware is non-portable and contains low level details: control and state registers, interrupt priorities, DMA starting address,…
- The kernel has several key components:
  - Device drivers are the hardware units managers; they hide the low level details.
  - File sys manager is the code that organises the data storage into files and directories, hiding low level details re disk blocks, for example.
  - Process management handles processes, allocating them resources and scheduling their execution.
  - Memory management is responsible for physical memory and virtual memory management.
  - Network services provide host-to-host and process-to-process communication across network.

Kernel services

- The kernel can be viewed as a collection of functions that user programs may call. They offer functionality and a higher level of abstraction of the computer.
- The repertoire of commands supported by the kernel defines the “virtual machine” which is platform-independent.
- To enter the kernel, the user program makes a *system call* by executing a *trap instruction* of some sort.
- This instruction switches the processor into a privileged operating mode (*kernel mode*) and jumps into the kernel through a well-defined trap address.
- Parameters passed with the trap instruction tell the kernel what service is requested.
- When the function is completed, the processor flips back to its normal operating mode (*user mode*) and returns control to the user program.

Trap instructions

- There are functions that require specific knowledge of handling resources – control registers, state register, sequence of operations, and a certain degree of protection – resources shared by several users/programs.
- These functions are coded as service routines; they are also known as system calls.
- The sequence of steps for a system call is:
  - System call is invoked by the user program;
  - OS function is performed;
  - Control returns to the user program
- Trap instructions are used to implement system calls.



TRAP Mechanism

- *A set of service routines.*
  - part of the kernel -- routines start at arbitrary addresses up to 256 routines
- *Table of service routines starting addresses.*
  - stored at x0000 through x00FF in memory
  - called System Control Block in some architectures
- *TRAP instruction.*
  - used by program to transfer control to operating system
  - 8-bit trap vector names one of the 256 service routines
- *Return to the user program.*
  - execution of user program will resume immediately after the TRAP instruction. Generally, the return address is saved in a CPU general register.

Library functions

- User programs have access to libraries and include in their code functions of these libraries – linked in the executables.
- Some library functions use system calls. The function itself parcels up the parameters correctly and then performs the trap.
- The function works as a wrapper for the system call.
- Example:

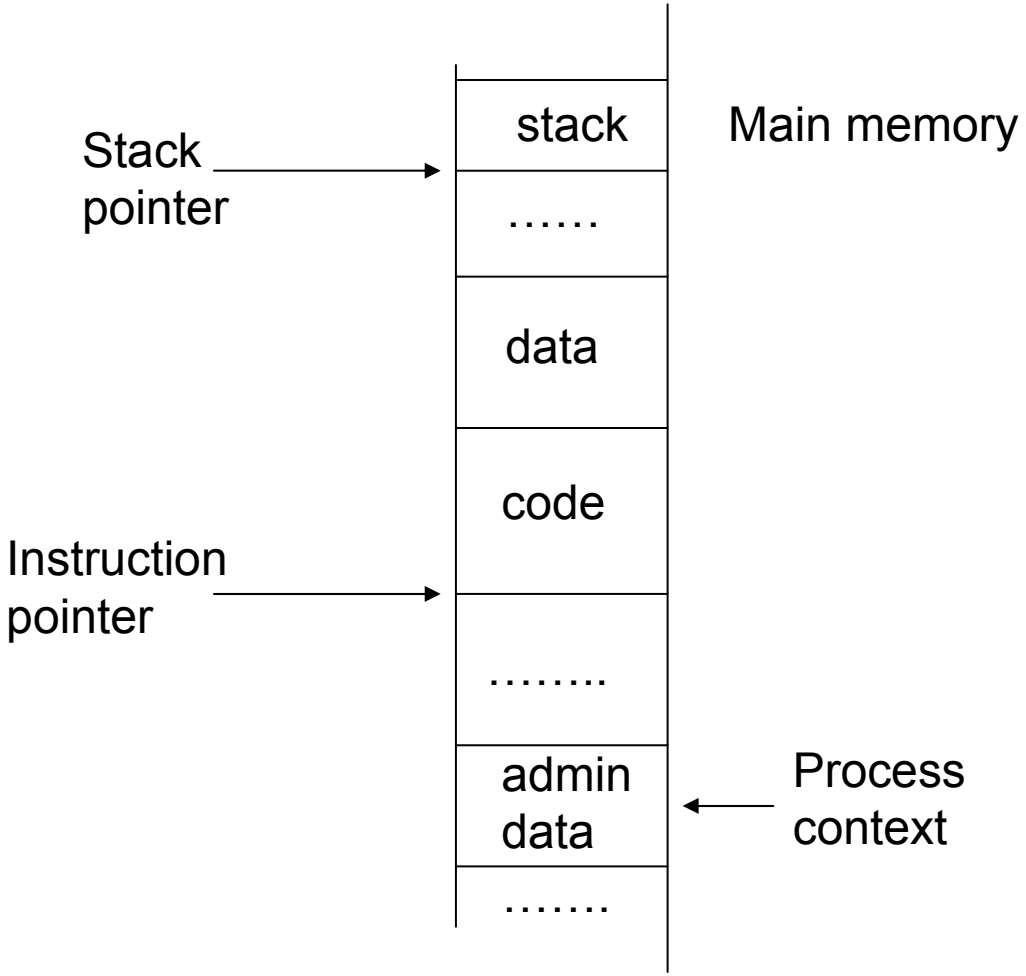
printf(“hello world”); → write(1, “hello world”, 11);

OS classes

- The large range of computing devices requires customized OS.
- General purpose computers run powerful OS: Unix, Linux, Windows,...
- Mobile devices, such as smart phones or sensors, run OS concerned with power saving: Symbian, iPhone OS, Android, TinyOS,...
- Embedded systems run scaled down versions of OS, event-driven.

Process

- Process definitions:
  - an instance of a running program;
  - a process is the context associated with a program in execution.
- The context represents state information:
  - program variables/values, stored in the user space;
  - management information such as process ID, priority, owner, current directory, open file descriptors, etc. stored in the kernel space
- The process consists of the executable (instructions), its data and administrative (management) information.



Process representation in the main memory.

Main components of process context

- Process ID = unique integer value
- Parent process ID
- Real user ID = the id of the user who started this process
- Effective user ID = user whose rights are carried (normally the same as above)
- Current directory = the start directory for looking up relative pathnames
- File descriptor table = table with data about all input/output streams opened by the process. It is indexed by an integer value called file descriptor.
- The environment = list of strings VARIABLE = VALUE used to customize the behaviour of certain programs.
- Code area
- Data area
- Stack
- Heap
- Priority
- Signal disposition = masks indicating which signals are awaiting delivery, which are blocked.
- umask = mask value used to ensure that specified access permissions are not granted when this process creates a file.

Process management

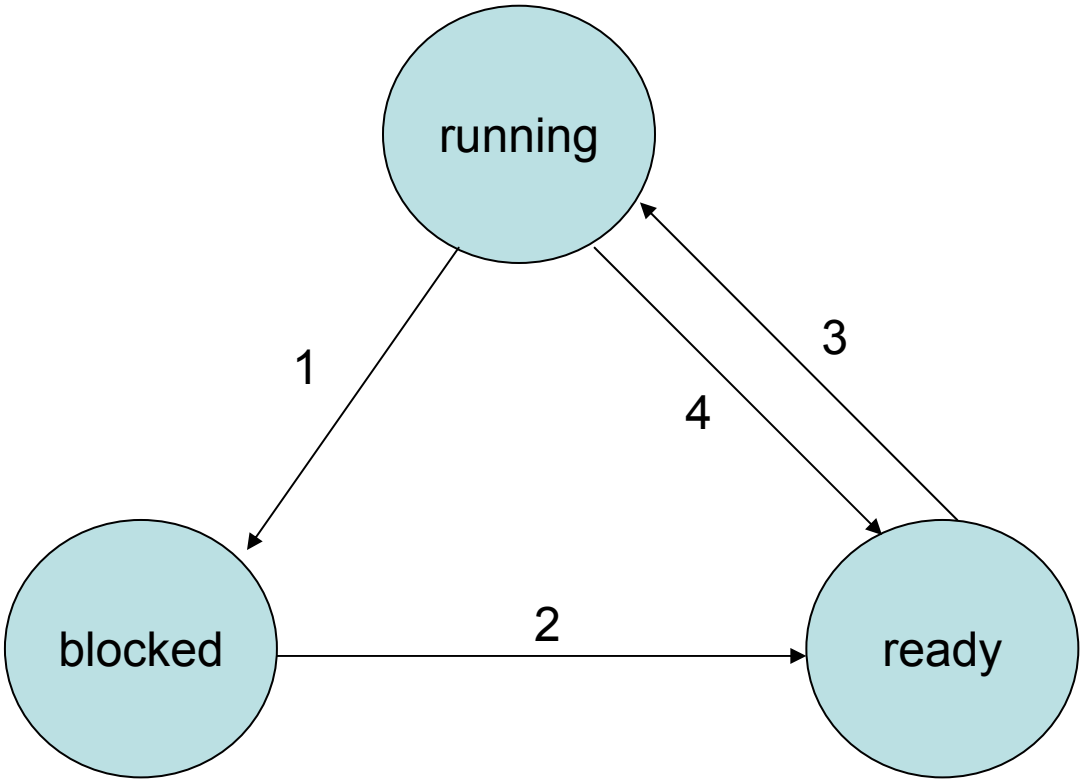
- *Create* – the internal representation of the process is created; initial resources are allocated; initialize the program that is to run the process.
- *Terminate* – release all resources; possibly inform other processes of the end of this one.
- *Change program* – the process replaces the program it is executing (by calling exec).
- *Set process parameters* – e.g., priority.
- *Get process parameters* – e.g., CPU time so far
- *Block* – wait an event, the completion of an I/O operation.
- *Awaken process* – after waiting, the process can run again.
- *Switch process* – process context switching.
- *Schedule process* – takes control of the CPU.

Child process

- A process can create a child process, identical to it, by calling fork() – Unix function. As the kernel creates a copy of the caller, two processes will return from this call.
- The parent and the child will continue to execute asynchronously, competing for CPU time shares.
- Generally, users want the child to compute something different from the parent. The fork() returns the child ID to the parent, while it returns 0 to the child itself. For this reason, fork() is placed inside an *if* test.
- Example:

```
int i;
if (fork()) {   /* must be the parent */
        for (i=0; i<1000; i++)
                printf("\t\t\tParent %d\n", i);
        }
        else  {   /* must be the child */
                for (i=0; i<1000; i++)
                        printf("Child %d\n", i);
        }
```

- Question: in what order will the two strings be printed ?



Process states

Thread

- A *thread* is known as a lightweight process; within a process we can have one (process ≡ thread) or more threads.
- All threads share the process context, including code.
- The context private to each thread is represented by the registers file and stack, the priority and own id.
- Generally the thread switch within the process is handled by the thread library, without calling the kernel. It is very fast as thread context is minimum.
- When a process starts execution, a single thread is executed, which begins executing the main() function of the program. It will continue so until new threads are created:

thread_create(char *stack, int stack_size, void (*func)(), void *arg);

Advantages/disadvantages

- Threads provide concurrency in a program. This can be exploited by multicore computers.
- Concurrency corresponds to many programs internal structure.

- If a thread changes directory, all threads in the process see the new current directory.
- If a thread closes a file descriptor, it will be closed in all threads.
- If a thread calls exit(), the whole process, including all its threads, will terminate.
- If a thread is more time consuming than others, all other threads will starve of CPU time.

Course goals and methodology

- Goals
  - to learn how processes and threads are managed, including scheduling.
  - to learn memory, physical and virtual, management.
  - to learn about device drivers.
  - to learn about file management system.
- Methodology
  - attending the lectures.
  - carrying out the lab work and assigned work.
  - using recommended references to learn more about specific topics.

- Text book
  - Silberschatz, Galvin, Gagne: Operating Systems Concepts with Java, Int Student Edition, John Wiley &Sons, 2011, isbn: 978-0-470-39879-1.

Course philosophy:

Collaborative learning process

www.cs.ucc.ie/~grigoras/CS2506

Grading	Continuous assessment:		20%
	1. 10 Labs		
	2. In-class test		
	Written exam:	90 min	80%
Lecture	50 min + 5 min review & questions		
Contact	Office: G69, Western Gateway Bldg		
	Email: d.grigoras@cs.ucc.ie, Subject: CS2506		
	Office hours: by appointment		