Lecture 2

Processes, threads, scheduling

What is a process?

- Process definitions:
    1. an *instance of a running program*, or
    2. 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.
- A process performs a job. There are user processes and OS processes.

I. Process structure

- The process consists of the executable (instructions), its data, stack, other buffer memory and administrative information (management, part of the context).

- The administrative information is mostly stored in the kernel space.

- The process has a *life cycle*: there are several states and the process switches from one state to another executing kernel decisions that follow specific events.

Administration: process context

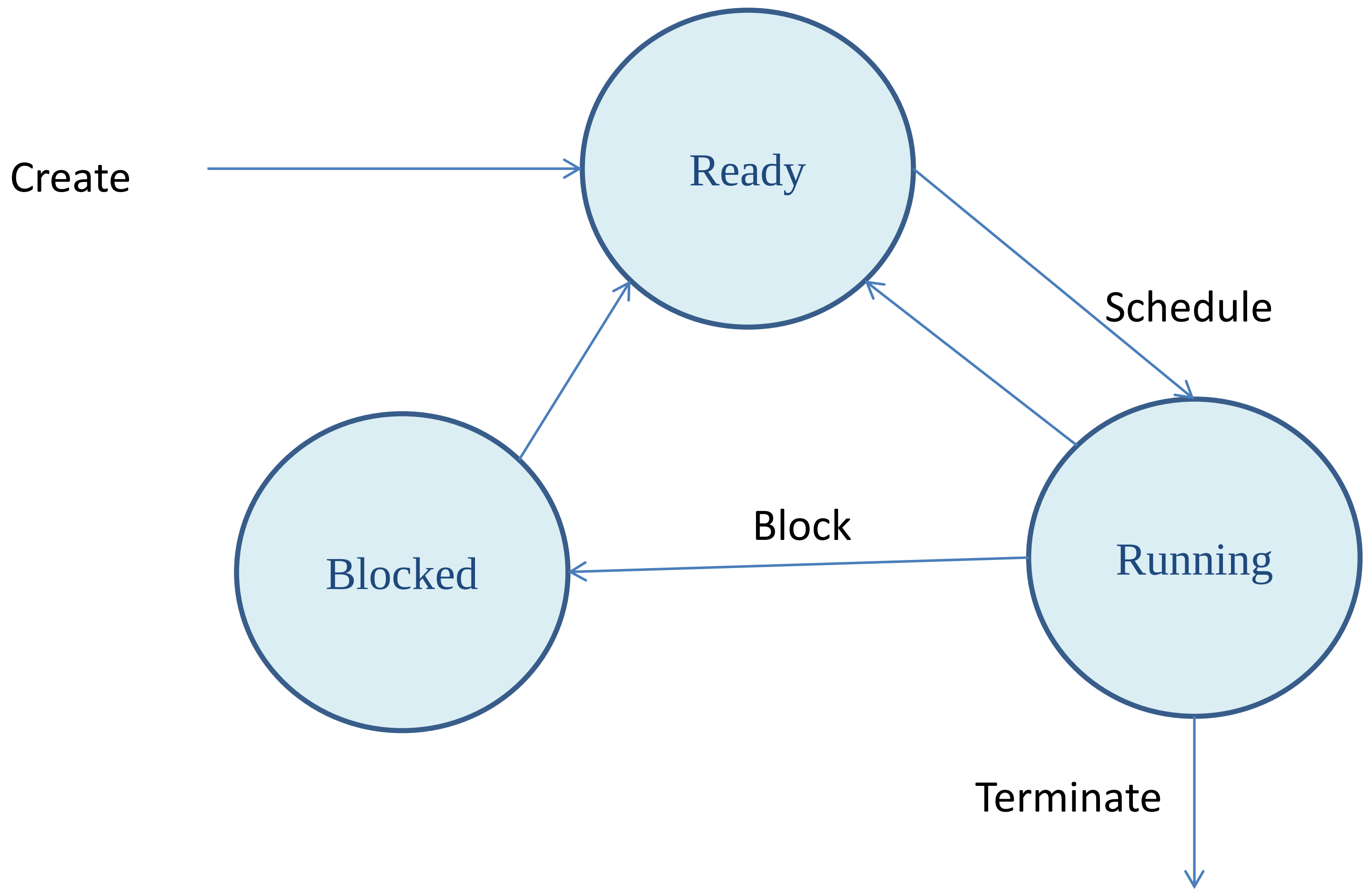
• Process ID:	unique integer value
• Parent process ID	
• Real user ID:	the id of the user who started this process
• Effective user ID:	user whose rights are carried (normally the same as above)
• Current directory:	the start directory for looking up relative pathnames
• File descriptor table:	table 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.
• 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

- Code area
- Data area
- Stack
- Heap

Process management operations

- 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 code it is executing (by calling the system call *exec*).
- Block: wait an event, e.g., the completion of an I/O operation.
- Awaken process: after sleeping, the process can run again.
- Switch process: process context switching.
- Schedule process: takes control of the CPU.
- Set process parameters: e.g., priority.
- Get process parameters: e.g., CPU time so far.

Basic process states



Create a child process

- A process can create a child process, identical to it, for example, 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 genealogy

- All processes are descendents of the *init* process, whose PID is 1.
- The kernel starts init in the last step of the boot process.
- The init process, in turn, reads the system init-scripts and executes more programs, eventually completing the boot process.

- Every process in the system has exactly one parent.
- Likewise, every process has zero or more children.
- Processes that are all direct children of the same parent are called siblings.

II. 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 and variables.
- The context private to each thread is represented by the registers file and stack, the priority and own id.
- When a process starts execution, a single thread is executed. It will continue so until new threads are created:

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

- Python has two libraries for creating processes and threads, multiprocessing and threading.

```
thread1 = threading.Thread(target=func, args(number,))
thread1.start()
thread1.join()
```

- Generally, the thread switch within the process is handled by the thread library, without calling the kernel. It is very fast as the thread context is minimum.
- Thread *affinity* (or process affinity) indicates on which core the thread (or process) is allowed to run.
- For example, in Java, it’s the task of the JVM’s optimizer to ensure that *objects affine to one thread* are placed close to each other in memory to fit into one core’s cache, but place objects affine to different threads not too close to each other to avoid that they share a cache line – avoid collision.
- The system represents affinity with a bitmask called a processor affinity mask.

- Comment: *thread affinity* forces a thread to run on a specific subset of cores and should generally be avoided, because it can interfere with the scheduler's ability to schedule threads effectively across cores.

Advantages/disadvantages

- Threads provide concurrency in a program. This can be exploited by multi-core 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.

Conclusions

- Processes and threads are the main execution entities managed by the OS kernel that compete for execution time.
- Processes life cycle consists of several states; switching between two states is time consuming.
- The thread switch is much faster than the process switch but, generally, threads run on the same core – don’t take advantage of many cores.