OS accessible only via specified procedures (system calls)

Execution of a system call:

- OS reads call parameters and checks appropriate privileges
- OS executes requested function in Supervisor mode
- OS returns result

Informally: Process is a program in execution.

States of a process:

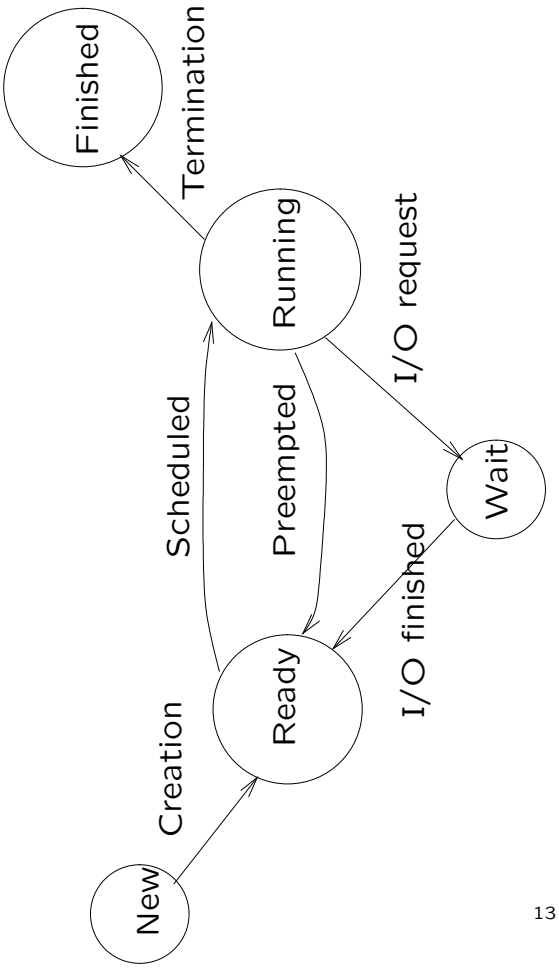
New Process is being created

Running Instructions are being executed

Waiting Process waiting for an event to occur (I/O; signal)

Ready Process waiting for processor only

Terminated Process has finished execution



OS allocates one Process Control Block (PCB) per process to store all relevant information, *e.g.*,

- Process state
- Program counter
- CPU-registers
- CPU-scheduling information
- Memory management information
- Accounting information
- I/O-status information

PCB used to identify process

Resource Queues

Ready-state associated with a queue of processes waiting to be executed
Scheduler picks one from the queue according to some criteria (later)

Instance of general principle:

Whenever we have a shared resource (here: processor), collect all waiting processes in a queue and use scheduler to decide who is next

Context Switches

Changing running process requires saving state of old one and loading state of new one
⇒ significant overhead involved (upto 1 ms).

Amount of work increases with complexity of OS

⇒ has become a real performance bottleneck

Will discuss one possible remedy (Threads) later

Process creation

Special system call for this (`fork` in UNIX)

Creating process is called parent, created process child.

Several possibilities for relation between parent and child:

- Concurrent execution vs. parent waiting for children
- Child is duplicate of parent or has separate code

Often all variants can be realised (UNIX).

Threads

Problem: Context switch takes too long
⇒ Can we reduce amount of material to be stored and re-fetched?

Solution: differentiate between

- data: memory, file pointers ···
- processor information: program counter, register set, stack

Processes manage all these data
introduce Threads, which are activity carriers together with processor information

Two possibilities for implementation of threads:

- User-level: OS doesn't know about them, only have a library. ⇒ Minimal overhead, but one blocked thread of process blocks whole process

- kernel-level: implementation done in OS: slower, but fairer allocation of resources possible

Some OS (e.g. Solaris) provide both versions.
Threads require good handling of concurrent processes.