# Processes, resources, threads

## Processes and resources

- A process is an elementary unit of activity managed by the operating system.
- Processes apply for access to resources which are needed to complete processing.
- Process = program at execution.
- A process consists of:
  - program code
  - data
  - resources like files, environmental variables
  - Process control block (PCB) — a description of the current processing state in the terms of OS's specific data structure
- A resource is a hardware or software component which is needed for processing
- Typically resources are defined at the level of the operating system rather than at the hardware level
- Examples of resources: processor, memory, file, etc.

## Managing processes and resources

- A manager is an abstraction (logically distinguished component) used to model OS working (thereby hide the details of specific resources or operations issued by the processes).
  - Process Manager — controls process states for safe and efficient resource utilization.
  - Resource Manager — allocates resources to processes following their requests, present state of the system and OS's allocation policy.

# OS’s data structures — descriptors

- To implement the process model, the operating system maintains a table, called the process table, with one entry per process. Each of those entries is called process descriptor, process control block, or PCB.
- This entry contains important information about the process’ state, including its program counter, stack pointer, memory allocation, the status of its open files, its accounting and scheduling information, and everything else about the process that must be saved when the process is switched from running to ready or blocked state so that it can be restarted later as if it had never been stopped.
- A counterpart of PCB for a general resource is the resource descriptor. Resource descriptor stores information about resource state including resource availability and allocation state.

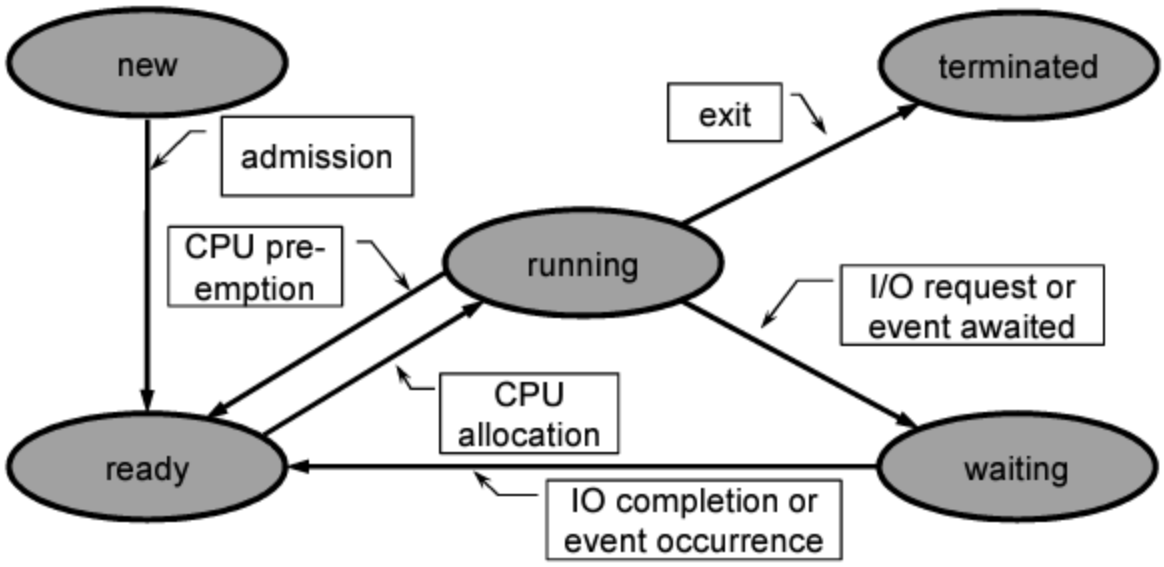
# Process descriptor

- process identifier(s)
- process state (ready or waiting etc.)
- owner identifier
- parent process identifier
- list of allocated resources (open files, sockets etc.)
- CPU registers contents — for context switch (to save/restore the state)
- permissions granted
- memory management information
- scheduling information (e.g. priority)
- pointers to scheduling queues
- accounting information

# Process states

- As a process executes, it changes state. The state of a process is defined in part by the current activity of that process. A process may be in one of the following states:
  - New — the process is being created
  - Running — instructions are being executed
  - Waiting — the process is waiting for some event to occur
  - Ready — the process is waiting to be assigned to a processor
  - Terminated — the process has finished execution

# Process state cycle




# Resource descriptor

- resource identifier
- resource type
- creator identifier

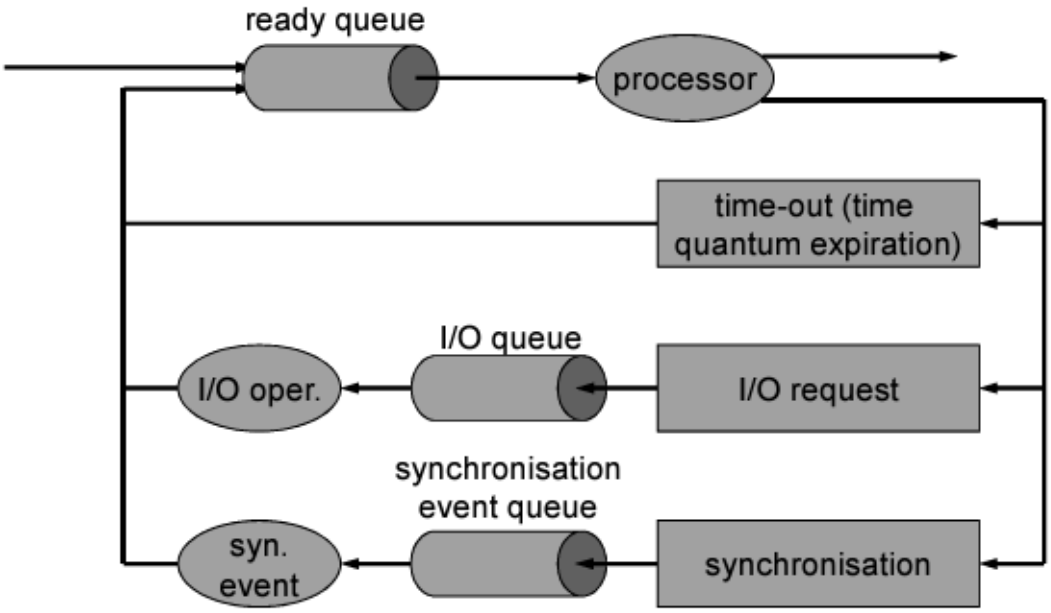
- list and count of resource units
- list (queue) of processes waiting for resource allocation
- procedure of resource allocation

# Resource classification

- With respect to the utilization
  - Reusable
  - Non-reusable
- With respect to the type of reusability
  - preemptable resource
  - non-preemptable resource
- With respect to how they are accesses
  - shared resource
  - exclusive resource

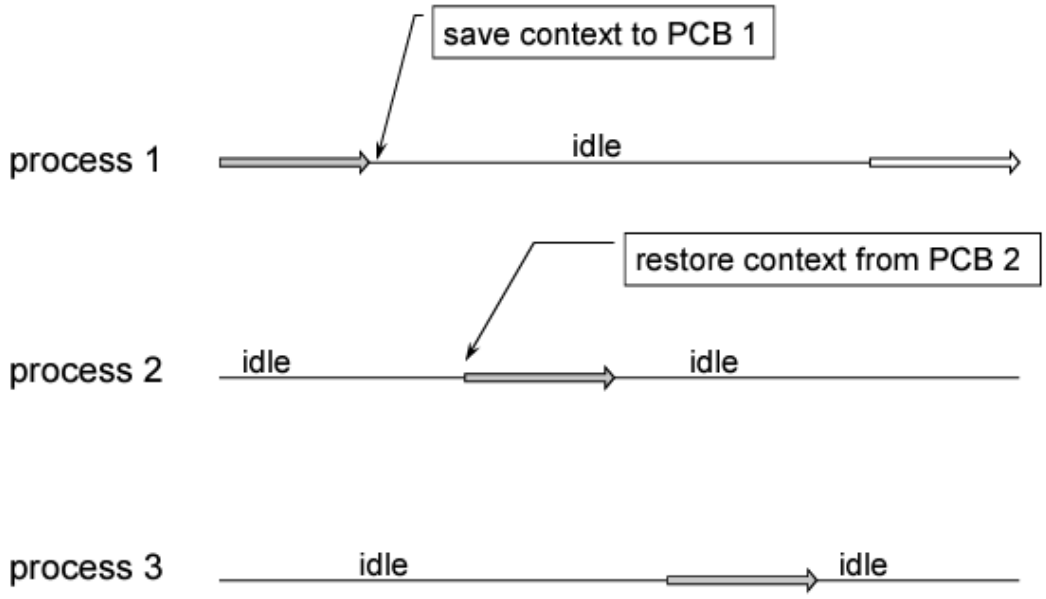
# Process queues

- The OS maintains several queues to organise process execution in a multitasking environment. Depending on the state, each process is one of the following queues:
  - job queue —all processes in the system,
  - ready queue — processes ready to be executed, residing in main memory,
  - I/O (device) queue — processes waiting for an I/O operation to complete
  - synchronisation event queue — processes waiting for synchronization with other processes.




# Context switch

- As we know, interrupts cause the OS to change a CPU core from its current task to run a kernel routine.
- When an interrupt occurs, the system needs to save the current context of the process running on the CPU core so that it can restore that context when its processing is done. The context is represented in the PCB of the process.
- Generically, we perform a state save of the current state of the CPU core, switch to another process, and then we perform a state restore to resume operations. This tasks is known as context switch.
- When a context switch occurs, the kernel saves the context of the old process in its PCB and loads the saved context of the new process scheduled to run.




# Schedulers

- Whenever the CPU becomes idle, the operating system must select one of the processes in the ready queue to be executed. The selection process is carried out by the CPU scheduler, which selects a process from the processes in memory that are ready to execute and allocates the CPU to that process.
- In many systems, this scheduling activity is broken down into three separate functions:
  - Short-term scheduler:

    Also known as the dispatcher, executes the most frequently and makes the fine-grained decision of which process to execute next. The short-term scheduler is invoked whenever an event occurs that may lead to blocking of the current process like signals, clock interrupts, I/O interrupts.
  - Medium-term scheduler:

    Medium-term scheduler is a part of the swapping function. Swapping involves moving part or all of a process from main memory to disk. When none of the processes in main memory is in the Ready state, the OS swaps one of the blocked processes out on to disk into a suspended queue.

Typically, the swapping-in decision is based on the need to manage the degree of multiprogramming.

- Long-term scheduler (job scheduler):

  The long-term scheduler determines which programs are admitted to the system for processing. Once admitted, a job or program becomes a process and is added to the queue for the short-term scheduler. In a batch system, newly submitted jobs are routed to disk and held in a batch queue. The long-term scheduler creates processes from the queue when it can. The long-term scheduler controls the number processes to balance system utilization.

# Threads

- Thread (light-weight process — LWP) is an object in a process, with a separate flow of control, sharing most resources allocated to the process with other threads of the process.
- A thread is a basic unit of CPU utilization; it comprises a thread ID, a program counter (PC), a register set, and a stack.
- It shares with other threads belonging to the same process its code section, data section, and other operating-system resources, such as open files and signals.
- Threads can be implemented in two ways:
  - Kernel-level threads — threads that are managed directly by the OS. The OS kernel maintains appropriate data structures to manage threads.

    The control block of the thread is kept by the OS kernel. It contains the value of the program counter, the contents of the CPU registers, the contents of the registers for stack management. The context switch between threads is performed by the OS, it is more expensive, but is allows more fair CPU allocation.
  - User-level threads — threads that are managed without kernel support. Thread data structures are maintained in the address space of the process, while kernel may not be even aware of them.

The user-level threads are maintained by the user code, the context switch is implemented by the local, tailer procedures. Threads of the same process may be starved, whenever one of them is blocked.