CHAPTER 3

PROCESS OPERATIONS

Dr. Soha S. Zaghloul

LAYOUT

- Process Termination
- Suspend-Resume (Blocked-Ready)
- Changing Priority
- Block-Wait
- Process Dispatch
- Overall Stage Diagram

PROCESS OPERATIONS

OSs should be able to perform principal operations on processes such as:

- Create a process
- Destroy (kill) a process
- Suspend a process
- Resume a process
- Change the priority of a process
- Block a process
- Wake up a process
- Dispatch a process
- Enable a process to communicate with other processes: this is known as Inter-Process Communication (IPC)

PROCESS TERMINATION (1) – EXIT()

A process terminates in one of two cases:

- The process finishes the execution of the last instruction.
- The process is forcibly terminated by another process.

As a process completes the execution of its last statement, it sends the *exit()* system call to the OS.

This causes the OS to delete the process: all its resources (memory, open files, and I/O buffers) are deallocated from the system.

PROCESS TERMINATION (2) – PARENT-CHILD RELATIONSHIP

A process can cause the termination of another process via a system call: *TerminateProcess()* in Windows.

Only the parent process can be invoked to terminate one of its child processes.

In order to do so, the parent process should know the identifier of the child to be deleted.

Therefore, it is important for a process to know the *pid* of the children processes and to save them in its PCB.

A parent may terminate the execution of one of its children for one of the following reasons:

- A child process exceeded the resources allocated for it. This is periodically inspected by the parent.
- The task assigned to the child is no more required.
- The parent exits: the OS doesn't allow a child process to continue without its parent.

Therefore, a parent terminates all its children processes before exiting. The termination is initiated by the OS: this is called **cascaded termination**.

PROCESS TERMINATION (3) – UNIX (1) – SYSTEM CALLS

In Unix, a process may be terminated directly using the *exit()* statement, or indirectly using the *return* statement in the *main()* program.

The *exit()* statement is provided with an exit status as a parameter.

The parent waits for the termination of a child. A parameter is returned to the parent with the status of the child termination as follows:

```
pid = wait(&status);
```

On the other hand, an *abort()* statement terminates a process forcibly.

PROCESS TERMINATION (4) – UNIX (2) – ZOMBIE VS. ORPHAN PROCESSES

Unix maintains a table of all active processes (parent & children) since their creation till their termination: this is called process table.

When a child process terminates, the following steps take place orderly:

- (1) The child process releases all allocated resources
- (2) The child process returns a status to the parent through the *wait()* statement issued by the parent
- (3) The child process identifier is deleted from the process table

If a child process terminates before returning its status to the parent *(because the parent didn't issue a wait() statement yet)*, then the child process is called a zombie process.

The *pid* of a zombie process remains in the process table until the parent issues a *wait()* statement to catch the status of its child before exiting.

Once the parent issues the *wait()* statement and receives the status of the zombie process, the *pid* of the latter is deleted from the process table.

On the other hand, if the parent process terminates without invoking a *wait()* statement, the zombie process is called an orphan process: a process with no parent.

The *init* process periodically checks for orphan processes. If found, the OS assigns *init* as an alternative parent.

Then, *init* invokes a wait statement to catch the exit status of the orphan child process, and deletes it permanently from the process table.

PROCESS OPERATIONS

OSs should be able to perform principal operations on processes such as:

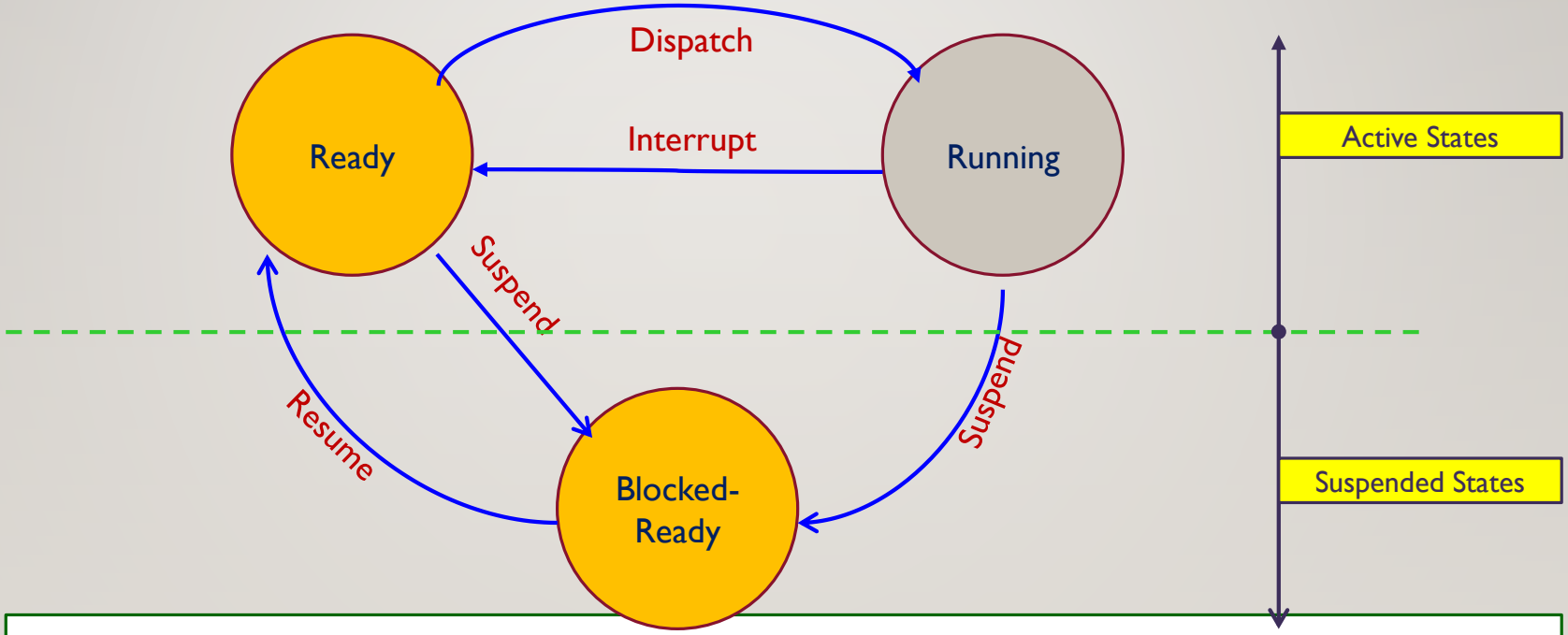
- Create a process
- Destroy (kill) a process
- Suspend a process
- Resume a process
- Change the priority of a process
- Block a process
- Wake up a process
- Dispatch a process
- Enable a process to communicate with other processes: this is known as Inter-Process Communication (IPC)

PROCESS SUSPEND/RESUME – BLOCKED-READY STATUS

Most OSs allow administrators, users or processes to suspend a process.

A suspended process is temporarily removed from the Ready queue without being destroyed. Therefore, it does no longer contend for CPU time.

This is useful for detecting security threats and for software debugging purposes.



A process may be blocked (suspended) while running.

Alternatively, a process may suspend another one in the ready queue or running on another processor.

In both cases, the blocked process returns to the Ready state when resumed.

PROCESS OPERATIONS

OSs should be able to perform principal operations on processes such as:

- Create a process
- Destroy (kill) a process
- Suspend a process
- Resume a process
- Change the priority of a process
- Block a process
- Wake up a process
- Dispatch a process
- Enable a process to communicate with other processes: this is known as Inter-Process Communication (IPC)

CHANGING THE PROCESS PRIORITY

The process priority is a value stored in one of the fields in the PCB.

Changing the priority of a process is applied by simply modifying the corresponding value in the process PCB.

This might accordingly change the order of the processes to be dispatched by the CPU scheduler: this depends on the design of the OS.

PROCESS OPERATIONS

OSs should be able to perform principal operations on processes such as:

- Create a process
- Destroy (kill) a process
- Suspend a process
- Resume a process
- Change the priority of a process
- Block a process
- Wake up a process
- Dispatch a process
- Enable a process to communicate with other processes: this is known as Inter-Process Communication (IPC)

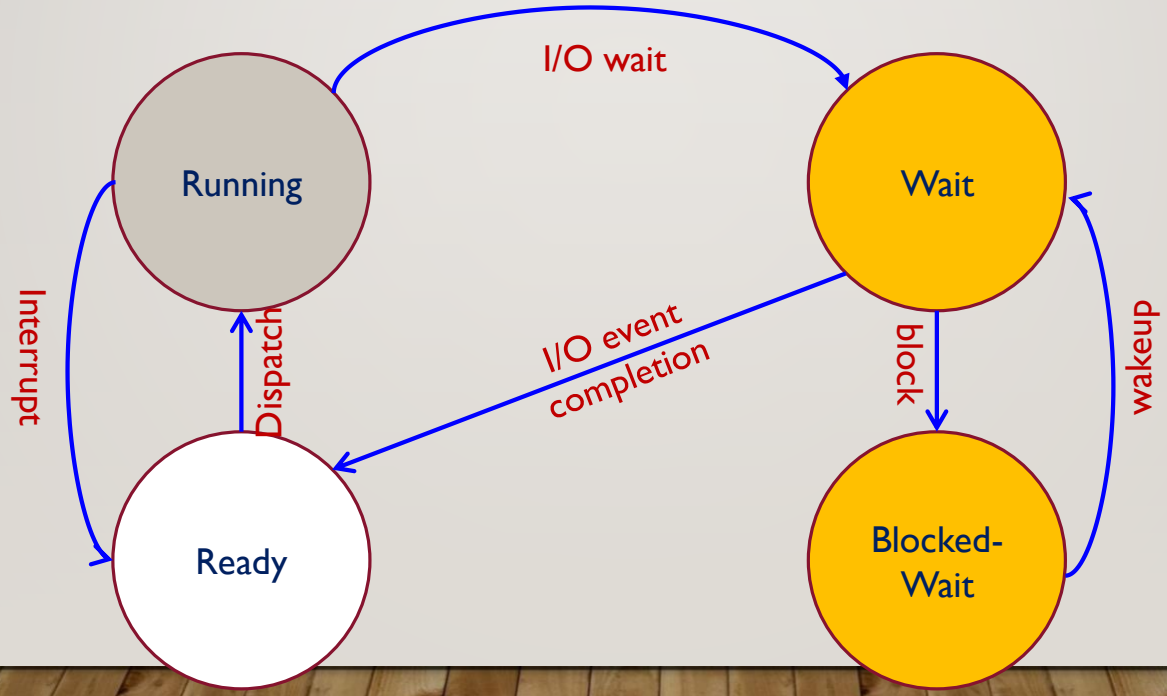
PROCESS BLOCK & WAKE-UP – BLOCKED-WAIT STATUS

A process might also be suspended (blocked) by another process while it is in the I/O device queue. Such state is called blocked-wait.

Such a process is temporarily removed from contention for I/O device queue without being destroyed.

This is useful for managing priorities for the usage of the I/O device.

When waked up, such process returns to the I/O device queue.



PROCESS OPERATIONS

OSs should be able to perform principal operations on processes such as:

- Create a process
- Destroy (kill) a process
- Suspend a process
- Resume a process
- Change the priority of a process
- Block a process
- Wake up a process
- Dispatch a process
- Enable a process to communicate with other processes: this is known as Inter-Process Communication (IPC)

PROCESS DISPATCH

As previously mentioned, a process that waits for the CPU assignment resides in the Ready Queue.

When the CPU Scheduler selects the process for execution, we say that the process is dispatched: ie. Is assigned to the CPU.

In a multiprocessing environment, a single ready queue is available to all processors.

Whenever a CPU becomes idle, the CPU scheduler looks for the processes in the Ready Queue and selects one of them to allocate the CPU.

Remember that OSs are designed to keep a balanced mix of I/O-bound and CPU-bound processes in order to make the system at its utmost efficiency.

OVERALL STATE DIAGRAM

The following figure depicts the overall state diagram of a process: