CHAPTER 3

SCHEDULING QUEUES

Dr. Soha S. Zaghloul

LAYOUT

- Job Queue
- Ready Queue
- Device Queue
- Queueing Diagram

INTRODUCTION

In a multiprocessor environment, our objective is to maximize the CPU utilization.

This is achieved as follows:

- Assign the CPU to another process when the running one switches to the waiting state.
- Switch the CPU among processes in a regular fashion so that users can interact with each executing program (multitasking).
- Provide queues in which a running process can wait until an event takes place and it regains the CPU.

An OS provides the following queues in order to guarantee the maximum CPU utilization:

- The Job Queue
- The Ready Queue
- The Device Queue

JOB QUEUE

For any program to execute, it must be loaded into the main memory.

Once a program is ready for execution, it looks for the main memory.

Therefore, all programs ready for execution are competing for the main memory.

The job scheduler, a routine in the OS, selects the program(s) that can be loaded into the main memory.

Such selection may be based on a variety of criteria such as:

- First-Come First-Served (FCFS)
- Priority
- Execution time
- Input/Output requirements

The job scheduler is also known as the long-term scheduler since it executes relatively infrequently.

Since the job scheduler reads from a disk, its execution time is relatively long.

READY QUEUE (1)

Once a program is selected by the job scheduler to be loaded into the main memory, it becomes a process.

All processes that compete for the CPU wait in the ready queue.

The CPU scheduler, a routine in the OS, assigns the CPU to a single process according to specific criteria.

Such selection may be based on a variety of criteria such as:

- First-Come First-Served (FCFS)
- Priority
- Execution time
- Input/Output requirements

The CPU scheduler is also called the short-term scheduler since it executes relatively frequently.

The CPU scheduler must be as fast as possible in order to make full use of the CPU.

READY QUEUE (2)

Therefore, a ready queue may be in one of the following situations:

- **Contains no processes**
  - No process is ready for execution.
  - Processes may not have been loaded in the memory yet, or
  - Processes may have been partially executed and are waiting in devices queues.
  - In such case, the CPU stays idle.
- **Contains a single process**
  - Once the running process deallocates the CPU (terminated/wait/ready) the process in the ready queue allocates the CPU.
  - When the running process encounters an I/O statement, it deallocates the CPU and waits in the device queue.
  - If no new process has arrived to the ready queue, then the CPU stays idle.
- **Contains multiple processes**
  - The CPU scheduler selects a process from the ready queue according to a specific criteria. The CPU is assigned to the selected process.
  - The CPU is kept busy as long as there are processes in the ready queue.

DEVICE QUEUE (1)

Each device in the system has a queue.

If a process needs a device, and the latter is busy, the process waits in the relevant queue.

Once the process is assigned the device, the former submits the job to be performed by the device, and is re-appended to the ready queue.

In other words, a device queue includes all processes that are waiting for a specific device (printer, disk drive, etc…) while it is busy processing another job.

DEVICE QUEUE (2) – SPOOLING

Spooling refers to storing processes in a buffer, a special area in memory, or on a disk where a device can access them when it is ready.

Therefore, some of these processes are spooled; ie. stored on the disk (job pool) where they are kept for later manipulation.

Spooling is necessary since different devices connected to the computer run at different rates.

The buffer provides a waiting station where data can reside while the slower devices catch up.

Spooling is the abbreviation of Simultaneous Peripheral Operations On-line. The term is originally invented by IBM in 1960 when they offered the machine IBM 7070.

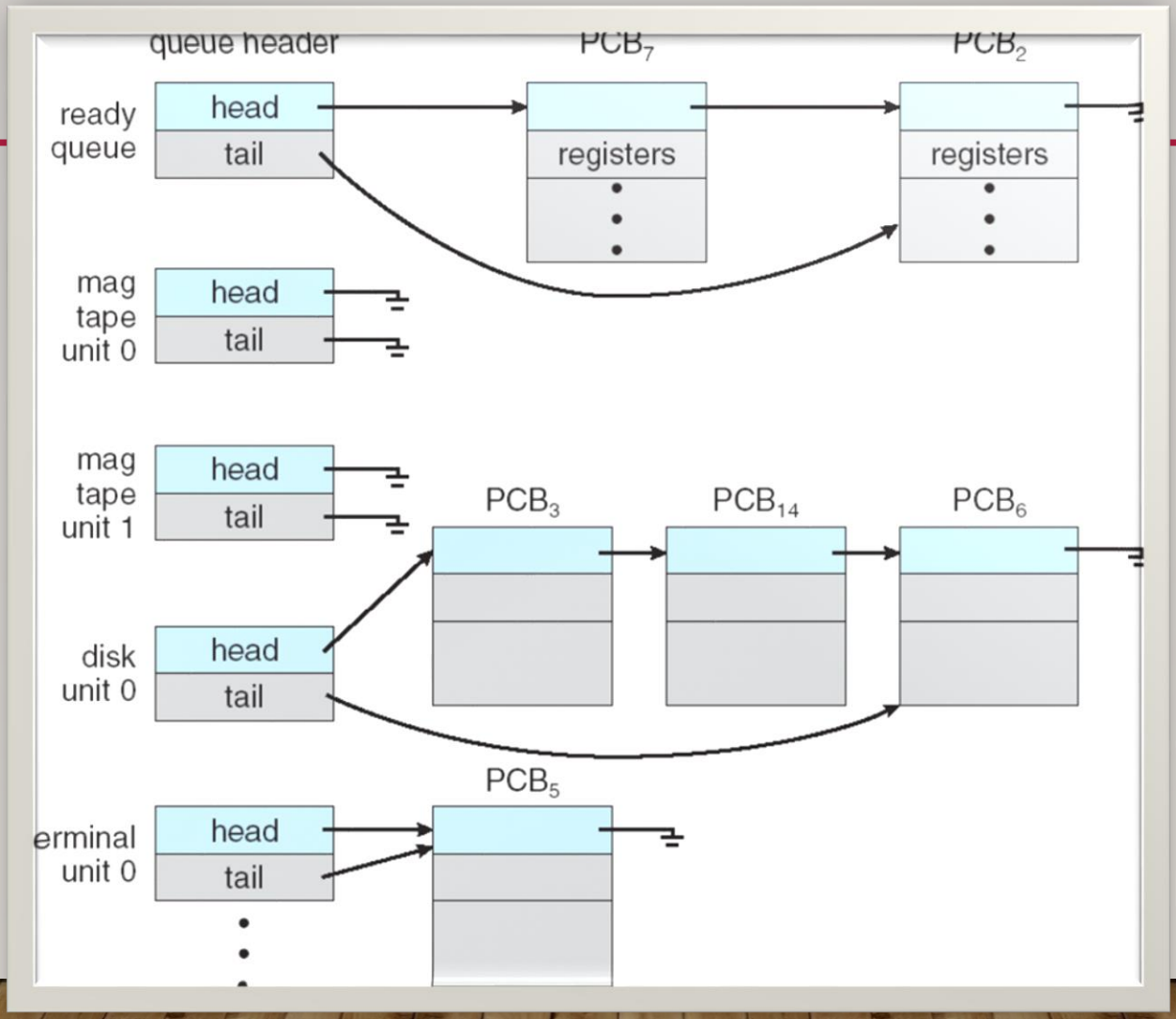
The most common application is print spooling where documents are loaded into a buffer (usually an area on a disk), and then the printer pulls them off the buffer at its own rate.

This allows you to use the computer while printing takes place in the background.

In addition, print spooling allows you to place a number of jobs in a queue to be printed, instead of waiting for each one before issuing the print command for the next one.

SCHEDULING QUEUES

The following figure depicts different scheduling queues in the computer system:

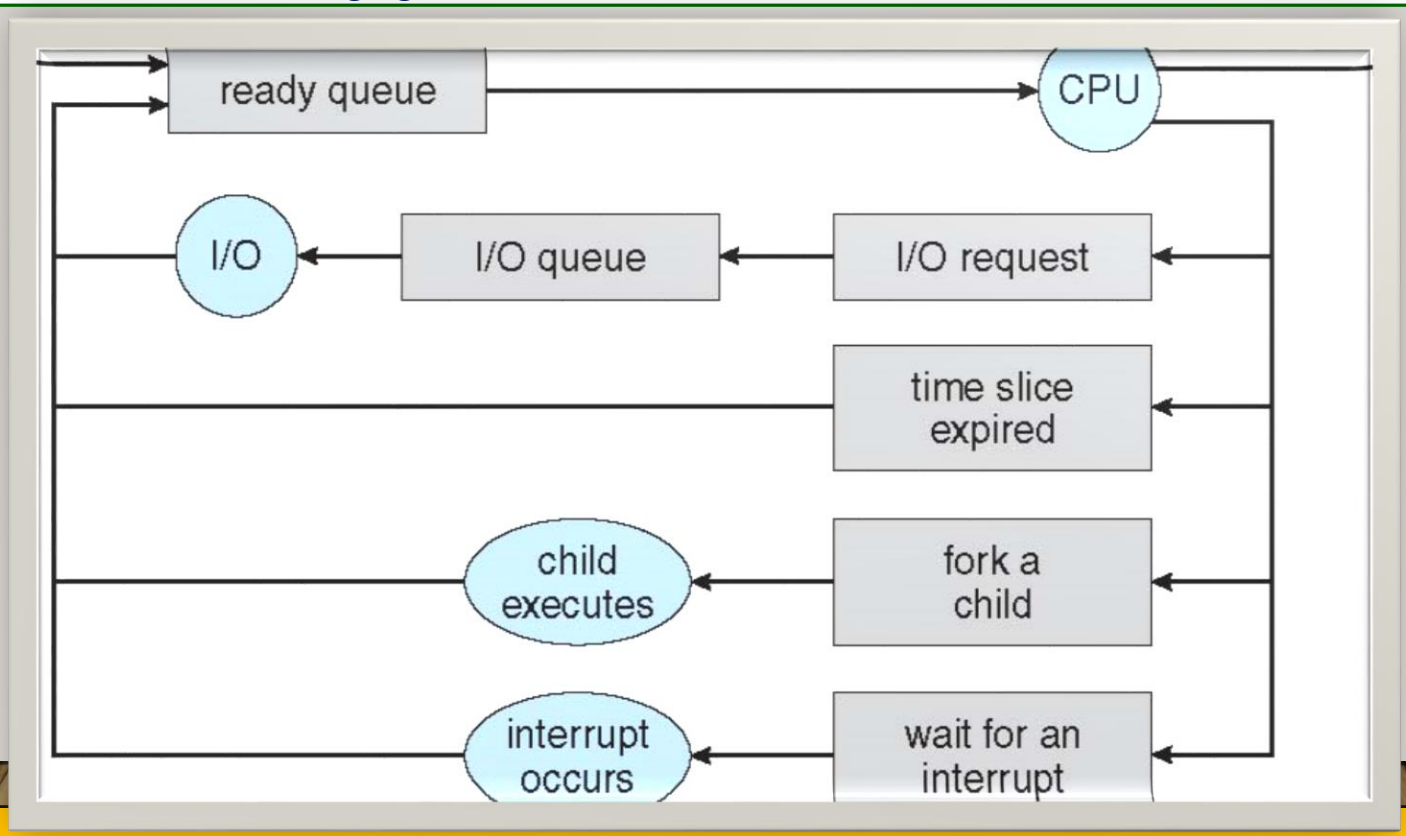


QUEUEING DIAGRAM (1)

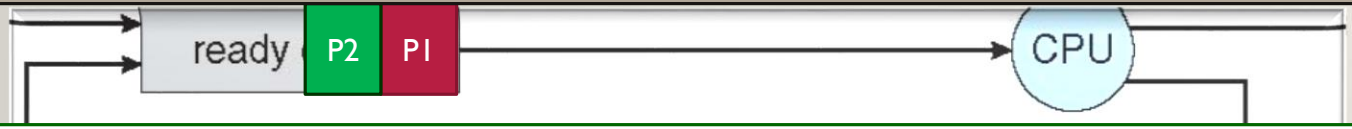
The queuing diagram is a common representation of process scheduling:

- Boxes represent a queue.
- Circles represent the resources that serve the queue.
- Arrows indicate the flow of the processes in the system.

Consider the following figure:



QUEUEING DIAGRAM (2) – PROCESS DISPATCH



Initially, a new process is added to the Ready Queue and waits there until selected for execution.

The process is selected for execution when:
(1) It is ready for execution AND
(2) It is selected for execution AND
(3) The CPU is available.
We say that the process is dispatched.

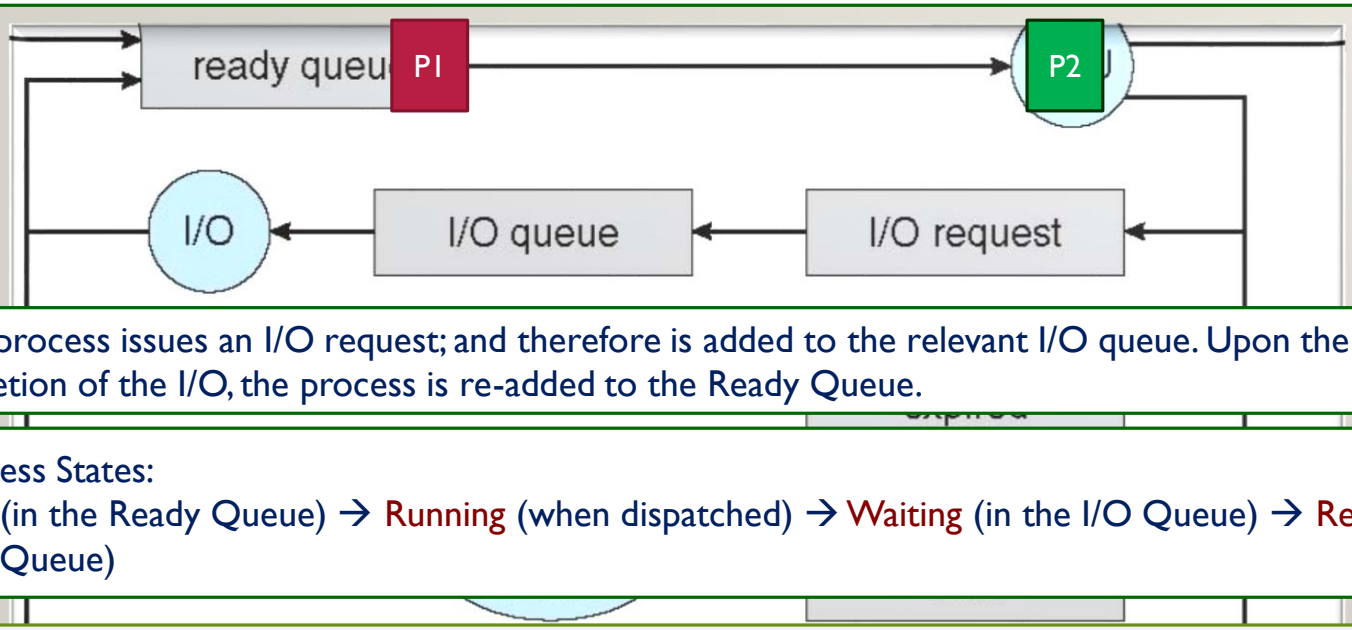
Process States:
Ready (in the Ready Queue) → Running (when dispatched)

Example: Assume P1 and P2 are in the ready queue →
State (P1) = Ready
State (P2) = Ready

According to the applied CPU Scheduler (may be P2 has higher priority), P2 is dispatched →
State (P1) = Ready
State (P2) = Running

QUEUEING DIAGRAM (3) – I/O REQUEST

Once the process is allocated the CPU and is executing, one of several events may occur:



~ The process issues an I/O request; and therefore is added to the relevant I/O queue. Upon the completion of the I/O, the process is re-added to the Ready Queue.

~ Process States:
Ready (in the Ready Queue) → Running (when dispatched) → Waiting (in the I/O Queue) → Ready (in the Ready Queue)

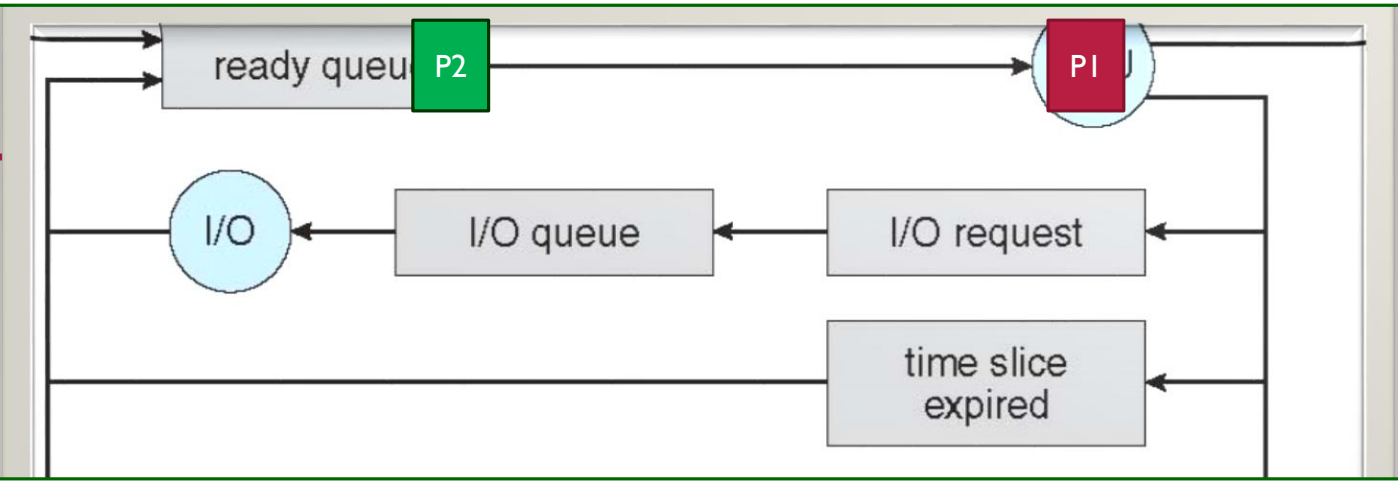
Example: Assume P2 issued an instruction to print a file → P2 is directed to wait in the Printer Queue until it becomes available for printing.
State (P2) = Waiting

In order to keep the CPU busy, P1 is dispatched:
State (P1) = Running

P2 is served by the printer. It submits the file to be printed to the device's buffer, and returns back to the Ready queue
State (P2) = Ready

QUEUEING DIAGRAM (4) – TIME SLICE EXPIRED

Once the process is allocated the CPU and is executing, one of several events may occur:



~ The time allocated for the process expires, so the OS invokes the CPU scheduler *(to be explained in more details later in this course)*

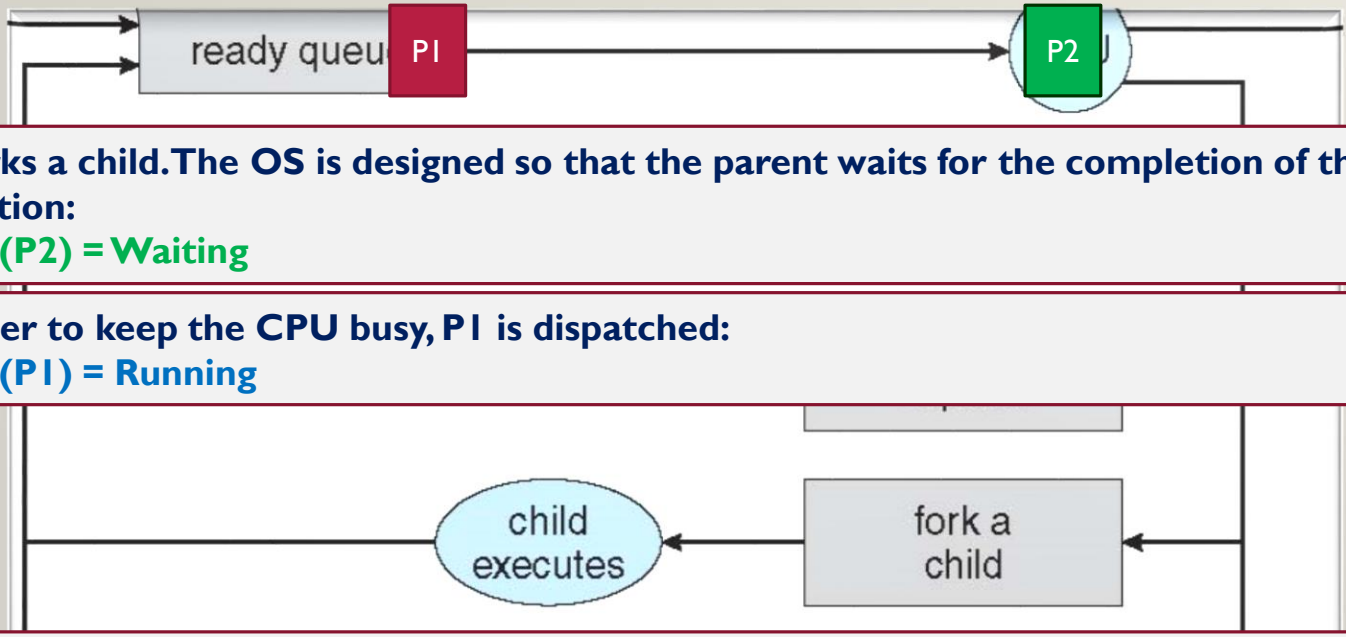
~ The CPU scheduler stops the running process, moves it to the end of the Ready Queue, and selects another process (if any) to be dispatched to keep the CPU busy.

~ Process States:
Ready (in the Ready Queue) → Running (when dispatched) → Ready (in the Ready Queue)

The OS is designed to allocate 5 ms to each process to run. P1 has no I/O command before that. So, after 5 ms, the CPU scheduler is invoked, P1 returns to the Ready Queue
State (P1) = Ready
In order to keep the CPU busy, P2 is dispatched:
State (P2) = Running

QUEUEING DIAGRAM (5) – FORK A CHILD

Once the process is allocated the CPU and is executing, one of several events may occur:



P2 forks a child. The OS is designed so that the parent waits for the completion of the child's execution:
State (P2) = Waiting

In order to keep the CPU busy, P1 is dispatched:
State (P1) = Running

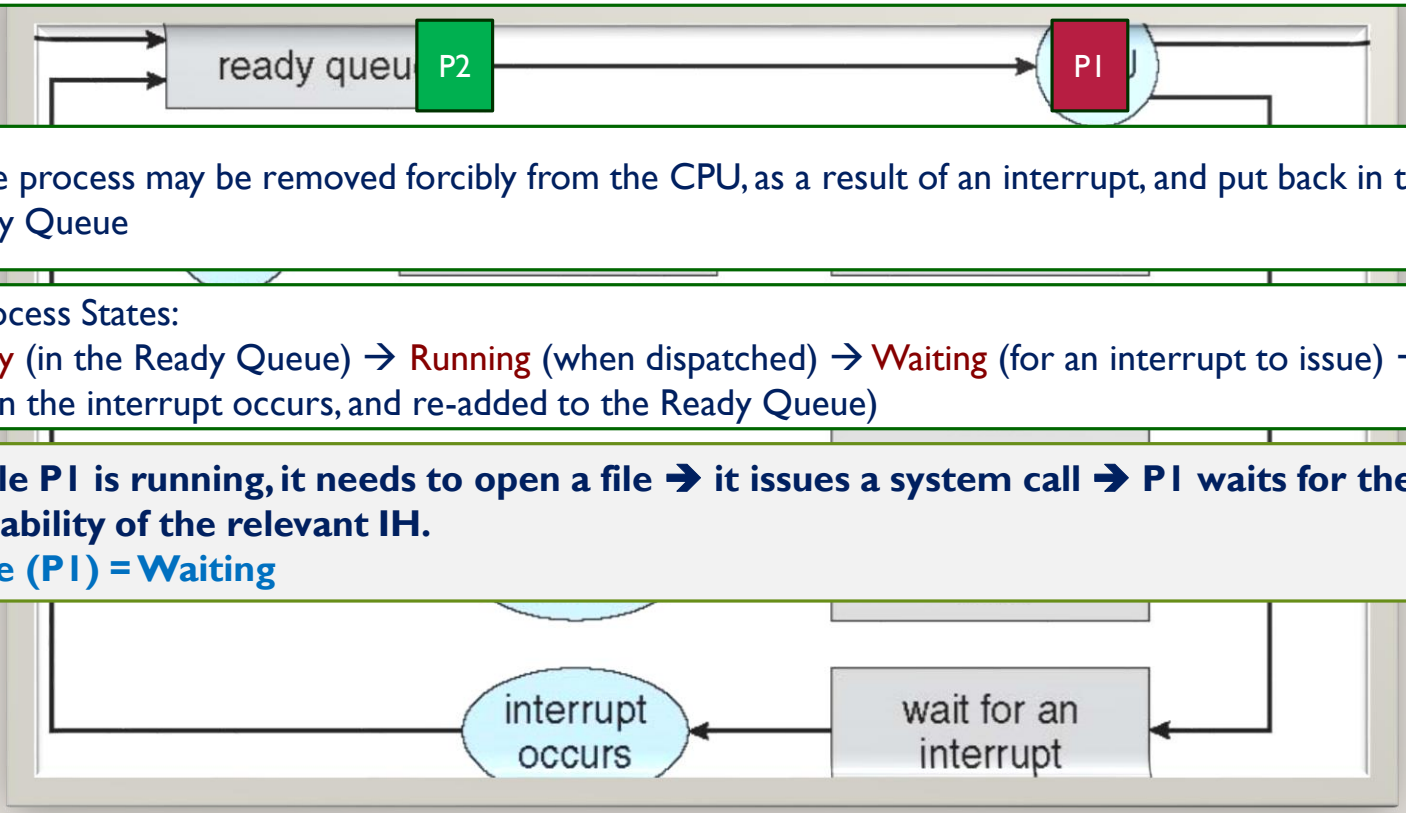
P2's child completes execution → P2 returns back to the Ready queue
State (P2) = Ready

~ The process may fork (spawn/create) a child. The process will have to wait until the child terminates *(this will be explained in more detail later in this course)*, then the process is re-added to the Ready Queue

~ Process States:
Ready (in the Ready Queue) → Running (when dispatched) → Waiting (till the child terminates) → Ready (in the Ready Queue)

QUEUEING DIAGRAM (6) – WAIT FOR AN INTERRUPT

Once the process is allocated the CPU and is executing, one of several events may occur:



~ The process may be removed forcibly from the CPU, as a result of an interrupt, and put back in the Ready Queue

~ Process States:
Ready (in the Ready Queue) → Running (when dispatched) → Waiting (for an interrupt to issue) → Ready (when the interrupt occurs, and re-added to the Ready Queue)

While P1 is running, it needs to open a file → it issues a system call → P1 waits for the availability of the relevant IH.
State (P1) = Waiting

The CPU switches to the kernel mode, then executes the Interrupt Handler
State (P2) = Ready

the interrupt handler completes → the CPU switches to the User Mode → P1 is returned back to the Ready Queue
State (P1) = Ready

QUEUEING DIAGRAM (7) – EXAMPLE 1: PERFORMING AN INPUT REQUEST

Assume ***P1*** is in the Running state.

P1 issues a read command from the keyboard *(the user should enter a value)*.

Therefore, the following steps take place:

- ***P1*** is appended to the keyboard queue ➔ state of ***P1*** = Waiting.
- To keep the CPU busy, another process ***P2*** is dispatched ➔ state of ***P2*** = Running.
- When the user enters the required value, the keyboard device controller sends an interrupt request to indicate that the input operation is complete. ***P1*** should move into the Ready queue.
- ***P1*** now waits for the availability of the relevant IH in a dedicated queue ➔ state of ***P1*** = Waiting (for an interrupt).
- As soon as the IH is available, ***P2*** is interrupted. The CPU now executes the IH.
- With the completion of the IH execution, ***P1*** is moved to the Ready queue ➔ state of ***P1*** = ready.
- The OS invokes the CPU scheduler to assign the CPU to one of the processes in the ready queue.

QUEUEING DIAGRAM (8) – EXAMPLE 2: PERFORMING AN OUTPUT REQUEST

Assume ***P1*** is in the Running state.

P1 issues a print command to the printer.

Therefore, the following steps take place:

- ***P1*** is appended to the printer queue ➔ state of ***P1*** = Waiting.
- To keep the CPU busy, another process ***P2*** is dispatched ➔ state of ***P2*** = Running.
- When ***P1*** submits the text to be printed, the printer device controller sends an interrupt request to indicate that ***P1*** should move into the Ready queue.
- ***P1*** now waits for the availability of the relevant IH in a dedicated queue ➔ state of ***P1*** = Waiting (for an interrupt).
- As soon as the IH is available, ***P2*** is interrupted. The CPU now executes the IH.
- With the completion of the IH execution, ***P1*** is moved to the Ready queue ➔ state of ***P1*** = ready.
- The OS invokes the CPU scheduler to assign the CPU to one of the processes in the ready queue.
- When the printer becomes available, its device controller sends another interrupt request to the OS to update it about its new status.
- The same steps mentioned above are repeated with respect to the running process.