# – Lab4 –

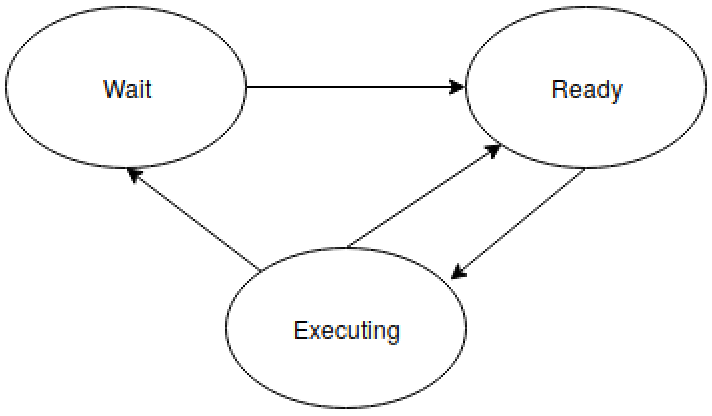
**Scheduling**

**- Investigating how to schedule tasks in an operating system**

## 1 Introduction

The purpose of this lab is to get an understanding on how different scheduling algorithms of an operating system can work. The lab is designed as a simulated environment, thus you will not work with real PCB's, but never the less, you will be able to get an understanding on how a real operating system determines which task is to be executed next. //Unsorted queues

This lab uses a three state model for executing tasks, described below:



The first state is called wait, which is a list of tasks that are not yet ready to be executed. The wait state is the default state for tasks read from a file. When the operating system clock reaches a value which is equal to the release time of a task, this task is moved from the waiting queue to the ready queue for execution. The scheduler then selects a task from the ready queue to be moved to the executing stage. When the task has finished executing, one of the following two operations will be done; if the task has finished all its time quanta, the task will be put in the waiting queue, if not the task will be moved back to the ready queue.

Our simulator has 3 base functions; **OS_wakeup_n**, **scheduler_n** and **dispatch_n**. The functionalities of the base functions are described in the list below:

1. OS_wakeup_n: This function is responsible for waking up tasks, i.e. comparing the release time of a task to the operating system clock (`OS_cycles`). If the value of the operating system clock is equal to the release time of the task, OS_wakeup_n will move this task from the waiting queue to the back of the ready queue.

2. scheduler_n: This function is responsible for determining which task should be run. The scheduler in our current implementation is very simple as it only support a Round Robin policy, this means the scheduler will return the first task of the ready queue and then exits. If the ready queue is empty, an `idle_task` will be returned, which is used for filling up empty time slots in the operating system.

3. dispatch_n: The dispatcher is responsible for executing the task returned from the scheduler. The current dispatcher will run the task for one time quanta and then move the task to the back of the ready queue.

### Time-frame

The deadline for completing this Lab for bonus points is wednesday **(for labs with Jakob) and monday (for labs with Sara), OBS!!!**, week 10.

## 2 Assignment

In this lab you will increase the functionalities of the scheduler by adding the option to schedule tasks according to SJF and Multiple Queues (MQ) with priorities.

1. Implement the Shortest Job first scheduling policy. In the scheduler code, you have a section dedicated to new scheduling policies. Implement a scheduling policy which re-organizes the ready queue in such a manner that the task with the earliest upcomming deadline is moved to the front of the ready queue.

2. Implement priority RR scheduling according to Multiple Queues. Use 3 queues, High priority queue with quantum 1, Medium priority queue with quantum 2 and Low priority with quantum 4. On activation, the process is placed last in the high priority queue. After executing its time quantum, the process will change to Medium priority. When the process has executed its medium quantum, the process will change to Low priority queue. When a process changes queue, it will be placed last in the new queue.