1. Differentiate Long term scheduler from a Short term scheduler.

ANSWER 1: Long term scheduler and short term scheduler are two types of schedulers that are used by the operating system to manage the processes. Here are some differences between them:

| **Long term scheduler** | **Short term scheduler** |
| --- | --- |
| Long term scheduler is also known as job scheduler[1](https://www.geeksforgeeks.org/difference-between-long-term-and-short-term-scheduler/)[2](https://www.geeksforgeeks.org/difference-between-short-term-medium-term-and-long-term-scheduler/). | [Short term scheduler is also known as CPU scheduler1](https://www.geeksforgeeks.org/difference-between-long-term-and-short-term-scheduler/)[2](https://www.geeksforgeeks.org/difference-between-short-term-medium-term-and-long-term-scheduler/). |
| Long term scheduler selects processes from the job pool, which is a collection of processes waiting for execution, and loads them into the ready queue, which is a queue of processes ready to run on the CPU[1](https://www.geeksforgeeks.org/difference-between-long-term-and-short-term-scheduler/)[2](https://www.geeksforgeeks.org/difference-between-short-term-medium-term-and-long-term-scheduler/). | Short term scheduler selects processes from the ready queue and allocates the CPU to one of them for execution[1](https://www.geeksforgeeks.org/difference-between-long-term-and-short-term-scheduler/)[2](https://www.geeksforgeeks.org/difference-between-short-term-medium-term-and-long-term-scheduler/). |
| Long term scheduler controls the degree of multiprogramming, which is the number of processes that can reside in the main memory at a time[1](https://www.geeksforgeeks.org/difference-between-long-term-and-short-term-scheduler/)[2](https://www.geeksforgeeks.org/difference-between-short-term-medium-term-and-long-term-scheduler/). | [Short term scheduler controls the degree of CPU utilization, which is the percentage of time that the CPU is busy executing processes1](https://www.geeksforgeeks.org/difference-between-long-term-and-short-term-scheduler/)[2](https://www.geeksforgeeks.org/difference-between-short-term-medium-term-and-long-term-scheduler/). |
| Long term scheduler runs less frequently than short term scheduler, as it only runs when there is enough space in the main memory or when the CPU utilization is low[1](https://www.geeksforgeeks.org/difference-between-long-term-and-short-term-scheduler/)[2](https://www.geeksforgeeks.org/difference-between-short-term-medium-term-and-long-term-scheduler/). | Short term scheduler runs more frequently than long term scheduler, as it runs whenever a process terminates or switches from running to waiting state, or when an interrupt occurs[1](https://www.geeksforgeeks.org/difference-between-long-term-and-short-term-scheduler/)[2](https://www.geeksforgeeks.org/difference-between-short-term-medium-term-and-long-term-scheduler/). |
| Long term scheduler affects the system performance and throughput, which is the number of processes completed per unit time[1](https://www.geeksforgeeks.org/difference-between-long-term-and-short-term-scheduler/)[2](https://www.geeksforgeeks.org/difference-between-short-term-medium-term-and-long-term-scheduler/). | [Short term scheduler affects the system response time and turnaround time, which are the measures of how fast a process gets executed](https://www.geeksforgeeks.org/difference-between-long-term-and-short-term-scheduler/) |

1. Write a simple program using fork() and wait() .

ANSWER 2: A simple program using fork() and wait() in C could look something like this:

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/wait.h>

int main()

{

pid\_t pid; // declare a variable to store the process id

pid = fork(); // create a child process

if (pid < 0) // check if fork() failed

{

printf("Fork error\n");

exit(1);

}

else if (pid == 0) // check if this is the child process

{

printf("Child: Hello, I am the child process with pid %d\n", getpid());

sleep(3); // simulate some work by the child

printf("Child: Goodbye, I am exiting now\n");

exit(0); // terminate the child process

}

else // this is the parent process

{

printf("Parent: Hello, I am the parent process with pid %d\n", getpid());

printf("Parent: I am waiting for my child to finish\n");

wait(NULL); // wait for the child process to terminate

printf("Parent: My child has terminated, I am exiting now\n");

exit(0); // terminate the parent process

}

}

Copy

This program creates a child process using fork() and then prints some messages from both the parent and the child. The parent process waits for the child to finish using wait(), which blocks the parent until the child exits or a signal is received[1](https://www.geeksforgeeks.org/wait-system-call-c/). The child process sleeps for 3 seconds and then exits normally. The exit status of the child is ignored by the parent, but it could be retrieved using a pointer argument to wait()[1](https://www.geeksforgeeks.org/wait-system-call-c/).

**The output of this program could look something like this:**

**Parent: Hello, I am the parent process with pid 1234**

**Parent: I am waiting for my child to finish**

**Child: Hello, I am the child process with pid 1235**

**Child: Goodbye, I am exiting now**

**Parent: My child has terminated, I am exiting no**

1. What is the need for a Medium-term scheduler ?

ANSWER 3: A medium-term scheduler is a component of an operating system that manages the execution of processes that are blocked or waiting for some event to occur. [It is also known as a swapping scheduler, because it can swap out processes from memory to free up space for other processes1](https://www.geeksforgeeks.org/medium-term-scheduler-in-operating-system/).

**Need for a Medium-term Scheduler**

The need for a medium-term scheduler arises from the following reasons:

* To control the degree of multiprogramming: Multiprogramming is the ability of an operating system to run multiple processes concurrently. However, too many processes in memory can cause excessive competition for CPU and other resources, leading to poor performance and low throughput. The medium-term scheduler can reduce the degree of multiprogramming by swapping out some processes that are not currently running or have low priority[1](https://www.geeksforgeeks.org/medium-term-scheduler-in-operating-system/)[2](https://www.geeksforgeeks.org/difference-between-long-term-and-medium-term-scheduler/).
* To improve resource utilization: By swapping out processes that are blocked or waiting, the medium-term scheduler can free up memory and other resources for other processes that are ready to run or have higher priority. This can improve the overall resource utilization and efficiency of the system[1](https://www.geeksforgeeks.org/medium-term-scheduler-in-operating-system/)[2](https://www.geeksforgeeks.org/difference-between-long-term-and-medium-term-scheduler/).
* To balance the process mix: The medium-term scheduler can also swap out processes based on their characteristics, such as CPU-bound or I/O-bound, long-term or short-term, interactive or batch, etc. This can help balance the process mix and ensure that different types of processes get a fair share of CPU and other resources[1](https://www.geeksforgeeks.org/medium-term-scheduler-in-operating-system/)[3](https://www.sarthaks.com/2416469/what-is-a-medium-term-scheduler).

**Functions of a Medium-term Scheduler**

The main functions of a medium-term scheduler are:

* Managing blocked or waiting processes: The medium-term scheduler decides which blocked or waiting processes should be swapped out from memory and which ones should be swapped in when they become ready to run. This may involve changing the priority or resource allocation of processes based on their current state or resource usage[1](https://www.geeksforgeeks.org/medium-term-scheduler-in-operating-system/)[3](https://www.sarthaks.com/2416469/what-is-a-medium-term-scheduler).
* Managing resource usage: The medium-term scheduler monitors the resource usage of different processes and adjusts the resource allocation accordingly. This may involve swapping out processes that are using too much memory or other resources, or swapping in processes that need more resources[1](https://www.geeksforgeeks.org/medium-term-scheduler-in-operating-system/)[3](https://www.sarthaks.com/2416469/what-is-a-medium-term-scheduler).
* Process prioritization: The medium-term scheduler prioritizes processes based on a predefined set of criteria and policies. This may involve giving preference to processes that are interactive, short-term, I/O-bound, etc., over processes that are batch, long-term, CPU-bound, etc.[1](https://www.geeksforgeeks.org/medium-term-scheduler-in-operating-system/)[3](https://www.sarthaks.com/2416469/what-is-a-medium-term-scheduler).

4. Where do the status of a Process gets stored ? Explain it with a diagram.

ANSWER 4: [The status of a process is stored in a data structure called a process control block (PCB), also sometimes called a process descriptor1](https://www.geeksforgeeks.org/process-table-and-process-control-block-pcb/). [The PCB contains all the information about a process, such as its identification, state, priority, registers, memory allocation, I/O devices, etc.1](https://www.geeksforgeeks.org/process-table-and-process-control-block-pcb/)[2](https://en.wikipedia.org/wiki/Process_control_block)

**Process Control Block**

[A PCB is created by the operating system when a process is initialized or installed, and it is updated whenever the process changes its state or performs any operation1](https://www.geeksforgeeks.org/process-table-and-process-control-block-pcb/). The PCB plays a key role in context switching, which is the process of switching from one process to another by saving and restoring the state of the processes[1](https://www.geeksforgeeks.org/process-table-and-process-control-block-pcb/)[2](https://en.wikipedia.org/wiki/Process_control_block).

The structure and content of a PCB may vary depending on the operating system and the hardware platform, but some common elements are[1](https://www.geeksforgeeks.org/process-table-and-process-control-block-pcb/)[2](https://en.wikipedia.org/wiki/Process_control_block):

* Process identification: This includes a unique identifier for the process (usually an integer) and other information such as the parent process ID, user ID, group ID, etc. The process ID is used to cross-reference other tables maintained by the operating system, such as memory tables, I/O tables, file tables, etc.
* Process state: This includes the current status of the process, such as new, ready, running, waiting or terminated. It also includes other information related to scheduling, such as priority value, CPU time used or remaining, waiting time, etc. The process state determines how the operating system manages the process and allocates resources to it.
* Process control: This includes information used by the operating system to control and manage the process itself. This may include pointers to other PCBs of related processes (such as children or siblings), signals or messages sent or received by the process, inter-process communication channels, etc. It may also include information about resource allocation and usage, such as memory address space, page tables, segment tables, open files list, I/O devices list, etc.
* CPU registers: This includes the content of general-purpose registers, program counter, stack pointer, frame pointer, status register, etc. These registers store the current context of the process when it is executing or suspended. They are saved and restored during context switching.

**Diagram**

A possible diagram of a PCB could look something like this:

+---------------------+

| Process ID |

| Parent ID |

| User ID |

| Group ID |

+---------------------+

| Process State |

| Priority |

| CPU Time |

| Waiting Time |

+---------------------+

| Child Pointer |

| Sibling Pointer |

| Signal List |

| Message Queue |

+---------------------+

| Memory Address |

| Page Table |

| Segment Table |

| Open Files List |

| I/O Devices List |

+---------------------+

| General Registers |

| Program Counter |

| Stack Pointer |

| Frame Pointer |

| Status Register |

+---------------------+

5.What are the various states a process can take over during its life time with a diagram.

ANSWER 5: A process is an instance of a program in execution. A process can take over different states during its lifetime, depending on its execution status and resource availability. [The operating system manages the process states and transitions between them1](https://www.geeksforgeeks.org/process-states-and-transitions-in-a-unix-process/)[2](https://www.geeksforgeeks.org/states-of-a-process-in-operating-systems/).

**Process States**

The common process states are:

* New: A process is in the new state when it is created by the operating system, but not yet ready to run. [It is the program that is present in secondary memory that will be picked up by the operating system to create the process1](https://www.geeksforgeeks.org/process-states-and-transitions-in-a-unix-process/)[3](https://www.javatpoint.com/os-process-states).
* Ready: A process is in the ready state when it is loaded into the main memory and waiting for the CPU to execute it. [Processes that are ready for execution by the CPU are maintained in a queue called the ready queue1](https://www.geeksforgeeks.org/process-states-and-transitions-in-a-unix-process/)[2](https://www.geeksforgeeks.org/states-of-a-process-in-operating-systems/)[3](https://www.javatpoint.com/os-process-states).
* Running: A process is in the running state when it is selected by the CPU scheduler and executing on one of the CPU cores. Only one process can be in the running state on a single core at a time[1](https://www.geeksforgeeks.org/process-states-and-transitions-in-a-unix-process/)[2](https://www.geeksforgeeks.org/states-of-a-process-in-operating-systems/)[3](https://www.javatpoint.com/os-process-states).
* Waiting or Blocked: A process is in the waiting or blocked state when it cannot continue executing until some event occurs, such as an I/O completion, a user input, a signal, a lock release, etc. The process remains in the main memory but does not use the CPU. Processes that are waiting for some event are maintained in a queue called the waiting queue[1](https://www.geeksforgeeks.org/process-states-and-transitions-in-a-unix-process/)[2](https://www.geeksforgeeks.org/states-of-a-process-in-operating-systems/)[3](https://www.javatpoint.com/os-process-states).
* Terminated or Completed: A process is in the terminated or completed state when it finishes its execution or is killed by the operating system or another process. The operating system deletes the process control block and releases the resources allocated to the process[1](https://www.geeksforgeeks.org/process-states-and-transitions-in-a-unix-process/)[2](https://www.geeksforgeeks.org/states-of-a-process-in-operating-systems/)[3](https://www.javatpoint.com/os-process-states).

**Process Transitions**

A process can move between different states based on its execution status and resource availability. Here are some examples of how a process can move between different states:

* New to Ready: When a process is created, it moves from the new state to the ready state when the operating system has allocated resources to it and it is ready to be executed[1](https://www.geeksforgeeks.org/process-states-and-transitions-in-a-unix-process/)[2](https://www.geeksforgeeks.org/states-of-a-process-in-operating-systems/).
* Ready to Running: When a process is selected by the CPU scheduler from the ready queue, it moves from the ready state to the running state and starts executing on one of the CPU cores[1](https://www.geeksforgeeks.org/process-states-and-transitions-in-a-unix-process/)[2](https://www.geeksforgeeks.org/states-of-a-process-in-operating-systems/).
* Running to Ready: When a process is preempted by the CPU scheduler due to a time slice expiration or a higher priority process arrival, it moves from the running state to the ready state and waits for another chance to execute[1](https://www.geeksforgeeks.org/process-states-and-transitions-in-a-unix-process/)[2](https://www.geeksforgeeks.org/states-of-a-process-in-operating-systems/).
* Running to Waiting: When a process requests an I/O operation or waits for some event to occur, it moves from the running state to the waiting state and releases the CPU[1](https://www.geeksforgeeks.org/process-states-and-transitions-in-a-unix-process/)[2](https://www.geeksforgeeks.org/states-of-a-process-in-operating-systems/).
* Waiting to Ready: When a process receives an I/O completion interrupt or a signal that indicates that the event it was waiting for has occurred, it moves from the waiting state to the ready state and becomes eligible for execution again[1](https://www.geeksforgeeks.org/process-states-and-transitions-in-a-unix-process/)[2](https://www.geeksforgeeks.org/states-of-a-process-in-operating-systems/).
* Running to Terminated: When a process completes its execution normally or exits with a return code, it moves from the running state to the terminated state and informs the operating system about its status[1](https://www.geeksforgeeks.org/process-states-and-transitions-in-a-unix-process/)[2](https://www.geeksforgeeks.org/states-of-a-process-in-operating-systems/).
* Waiting to Terminated: When a process is killed by another process or by an external signal while it is in the waiting state, it moves from the waiting state to the terminated state and informs the operating system about its status[1](https://www.geeksforgeeks.org/process-states-and-transitions-in-a-unix-process/)[2](https://www.geeksforgeeks.org/states-of-a-process-in-operating-systems/).

**Diagram**

A possible diagram of process states and transitions could look something like this:

+------+ create +-------+ dispatch +--------+ preempt +-------+

| New | -----------> | Ready | -----------> | Running| --------> | Ready |

+------+ +-------+ +--------+ +-------+

^ | ^ |

| | | |

| V | |

| +--------+ |

| | Waiting| |

| +--------+ |

| ^ |

| | |

+--------------------+--------------------+

|

V

+-----------+

| Terminated|

+-----------+