# Project:Shell

Task 1.5: Provide a concise and descriptive answer to the following questions.

Q1: What does the mmap() function do?

The linux manual will help with the basic definition of this function:

mmap, munmap - map or unmap files or devices into memory

The mmap() function creates a mapping in the virtual adress space of the process in which it is called.

The function takes the following parametres:

void \*mmap(void \*addr, size\_t length, int prot, int flags,

int fd, off\_t offset);

First of all, we see that this function is of type void – it doesn't return any parameter.

The addr argument specifies the starting adress where the new mapping will be created.

The length parameter contains the length of the new mapping – hence it must be greater than zero.

The prot argument can be used for selecting the desired memory protection of the mapping

It can have one or more of the following values / flags:

PROT\_EXEC: Pages may be executed.

PROT\_READ: Pages may be read.

PROT\_WRITE: Pages may be written.

PROT\_NONE: Pages may not be accessed.

The flags argument determines whether updates to the mapping are visible to other processes mapping the same region, and whether updates are carried through to the underlying file.

It can have one or more of the following values / flags:

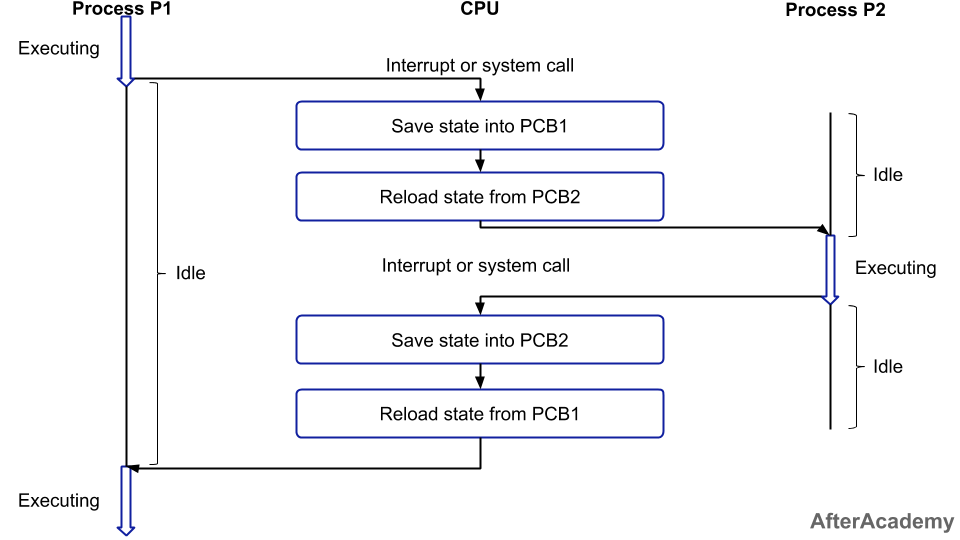
MAP\_SHARED, MAP\_SHARED\_VALIDATE,MAP\_PRIVATE, MAP\_32BIT, MAP\_ANON/ MAP\_ANONYMOUS

Q2: What happens during a context switch? Do we want the OS to perform many or few context switches? Explain.

A context switch is a process that involves switching of the CPU from one process or task to another. In this phenomenon, the execution of the process that is present in the running state is suspended by the kernel and another process that is present in the ready state is executed by the CPU.

Context is the contents of a CPU's registers and program counter(PC) at any point in time. Context switching can happen due to the following reasons:

* When a process of high priority comes in the ready state. In this case, the execution of the running process should be stopped and the higher priority process should be given the CPU for execution.
* When an interrupt occurs - the process in the running state should be stopped and the CPU should handle the interrupt before doing something else.
* When a transition between the user mode and kernel mode is required then context switching is required.



The picture above represents a context switch: Process P1 was running when a interrupt/system call happened, and the OS wants to run process P2 next. This could be due to the scheduler running different processes interchangeably - for some amount of time, or that some system call happened that result in the execution of process P2.

Also, from the picture we see the basics of what goes on during a context switch – the state of the running process is saved so that when the CPU runs it next time – it starts from where it was interrupted. Also, the state of the new process must be reloaded as one process can't run in the state of another process.

However, as context switching takes some amount of time to execute – it can be costly in terms of processor time. During this time, the CPU does no useful work – and in this way, context switching represents an overhead.

Having many / a few context switches is really a tradeoff between OS control and CPU utilization. If we perform a few context switches – we don't lose as much time, but our OS doesn't have nearly as much control.