Name: Duvvuri Srinath

Roll No: 21CS01018

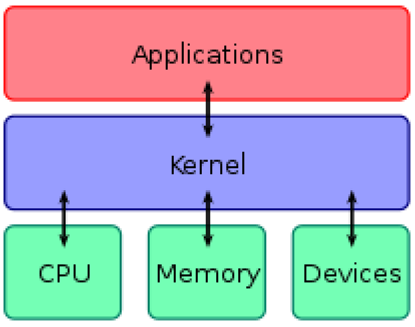
COA LAB ASSIGNMENT 2
Group – 13

Contents:

1. Kernel
2. Thread
3. Process
4. SIMD
5. GPU Memory Hierarchy
6. Scheduler
7. Warp
8. Thread Block

KERNEL:

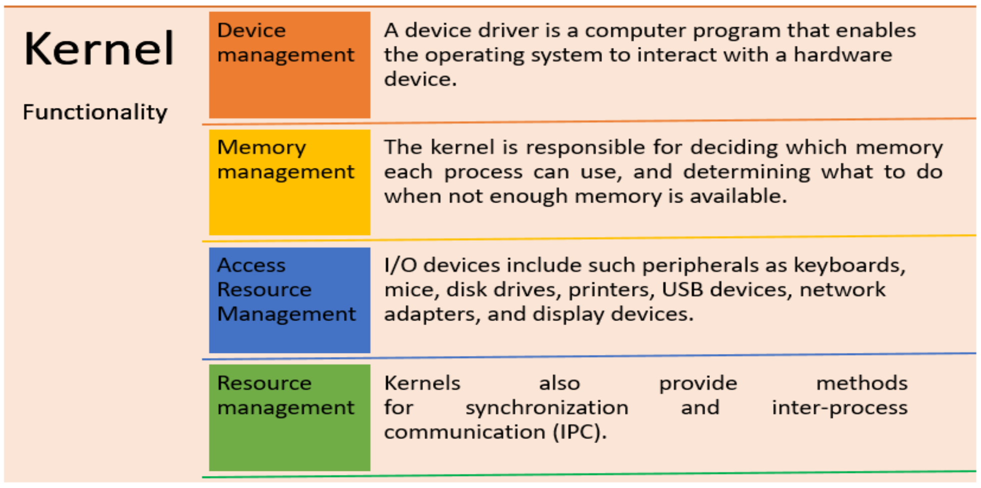
The kernel is a computer program at the core of a computer's operating system and generally **has complete control over everything** in the system.



Kernel acts as **bridge** between **application software** and **hardware** of computer

Objectives of Kernel:

Kernel is responsible for various tasks such as **disk management, task management** and **memory management**.



Kernel

Functionality

Device management	A device driver is a computer program that enables the operating system to interact with a hardware device.
Memory management	The kernel is responsible for deciding which memory each process can use, and determining what to do when not enough memory is available.
Access Resource Management	I/O devices include such peripherals as keyboards, mice, disk drives, printers, USB devices, network adapters, and display devices.
Resource management	Kernels also provide methods for synchronization and inter-process communication (IPC).

Types of Kernels:

1. **Monolithic Kernel:**
   In this type of kernel **same memory space is used** to implement user services and kernel services.
   Advantage: It leads to faster operations because there is less software involved as the memory is shared.
   Disadvantage: If the service generates any error, the whole system crashes down.
   Ex: Unix, Linux, Open-VMS etc.

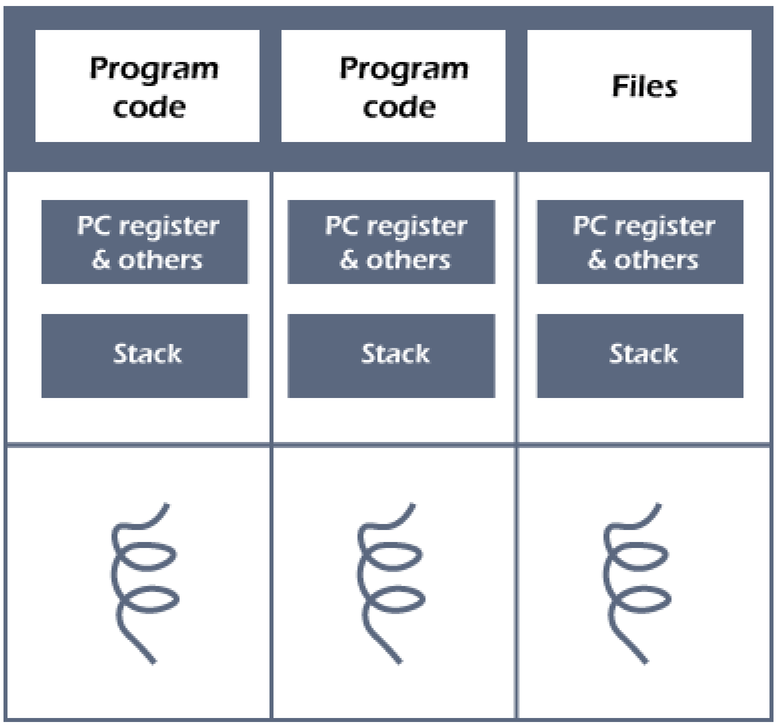
2. **Micro-Kernel:**
   This type of architecture **structures the operating system** by removing all the non-essential components from the kernel and implementing them as **system and user-level programs**.
   Advantage: It is easier to customize the operating system to meet specific requirements.
   Disadvantage: Message passing between user-level processes can be slower than direct system calls in monolithic kernel.
   Ex: Blackberry QNX, Genode, HelenOS, etc.

3. **Hybrid Kernel:**
   Hybrid kernels are also known as **modular kernels**, and it is the **combination of both Monolithic and Microkernels**.
   Advantage: It has the speed of monolithic kernels and the modularity of microkernels.
   The idea behind a hybrid kernel is to have a kernel structure similar to that of a microkernel, but to implement that structure in the manner of a monolithic kernel. In contrast to a microkernel, all (or nearly all) operating system services in a hybrid kernel are still in kernel space.
   Ex: Microsoft Windows NT Kernel

4. **Nano Kernel:**
   As the name suggests, *in Nanokernel, the complete code of the kernel is very small, which means the code executing in the privileged mode of the hardware is very small.* Here the term nano defines a kernel that supports a nanosecond clock resolution. This leads in providing hardware abstraction even with very small size.
   Ex: Eros etc.

Thread

Thread is a basic unit of CPU utilization. It consists of thread id, program counter, register set and stack. A system can have single-threaded or multi-threaded mechanism. Threads within a process share code, data, and files between them. They have their own stack and register set. A traditional or a heavyweight process could only perform a single task at a time. If a process has multiple threads of control then many tasks can be performed simultaneously and is highly optimised in comparison to the single thread process.



Three threads of same process

A thread can be formally defined as a single sequential flow of execution of tasks so it is also known as thread of execution or thread of control. Each thread of the same process makes use of a separate program counter and a stack of activation records and control blocks.

Advantages of multithreaded processes:

1. If the process is divided into multiple threads, if one thread completes its execution, then its output can be immediately returned.

2.Multi-threaded programming ensures effective utilisation of the multiprocessor system effectively for faster execution of the process.

3. Communication between multiple threads is easier, as the threads share a common address space.

Types of threads:

1. User-level threads:

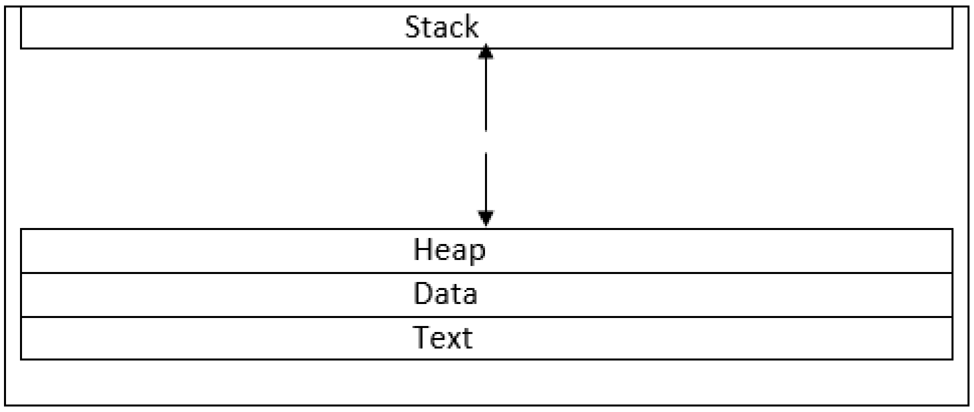
User Level Thread is a type of thread that is not created using system calls. The kernel has no work in the management of user-level threads. User-level threads can be easily implemented by the user.

2.Kernel-level threads:

A kernel Level Thread is a type of thread that can recognize the Operating system easily. Kernel Level Threads has its own thread table where it keeps track of the system. The operating System Kernel helps in managing threads.

PROCESS

A process is basically a program in execution which happens in a sequential manner. A program becomes a process when it is loaded into the memory. It can be divided into four sections- stack, heap , data ,text.



Process Diagram

Stack

The process stack stores temporary information such as method or function arguments, the return address, and local variables.

Heap

This is the memory where a process is dynamically allotted while it is running.

Text

This consists of the information stored in the processor's registers as well as the most recent activity indicated by the program counter's value.

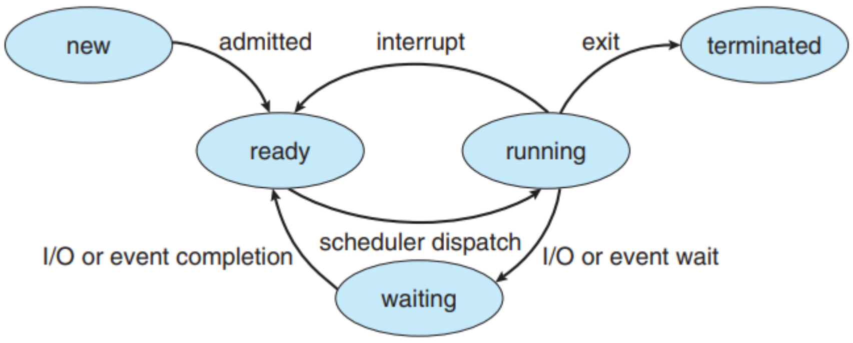
Data

This section contains the global and the static variables.

Process Life Cycle

When a process executes, it passes through different states.

In general, a process can have one of the following five states at a time.



- New. The process is being created.
- Running. Instructions are being executed.
- Waiting. The process is waiting for some event to occur (such as an I/O completion or reception of a signal).
- Ready. The process is waiting to be assigned to a processor.
- Terminated. The process has finished execution.

A PCB contains the following information:

1.**Process state**. The state may be new, ready, running, waiting, halted, and so on.

2. **Program counter**. The counter indicates the address of the next instruction to be executed for this process.

3. **CPU registers**. The registers vary in number and type, depending on the computer architecture. They include accumulators, index registers, stack pointers, and general-purpose registers, plus any condition-code information. Along with the program counter, this state information must be saved when an interrupt occurs, to allow the process to be continued correctly afterward

4. **CPU-scheduling information**. This information includes a process priority, pointers to scheduling queues, and any other scheduling parameters

5. **Memory-management information**. This information may include such items as the value of the base and limit registers and the page tables, or the segment tables, depending on the memory system used by the operating system.

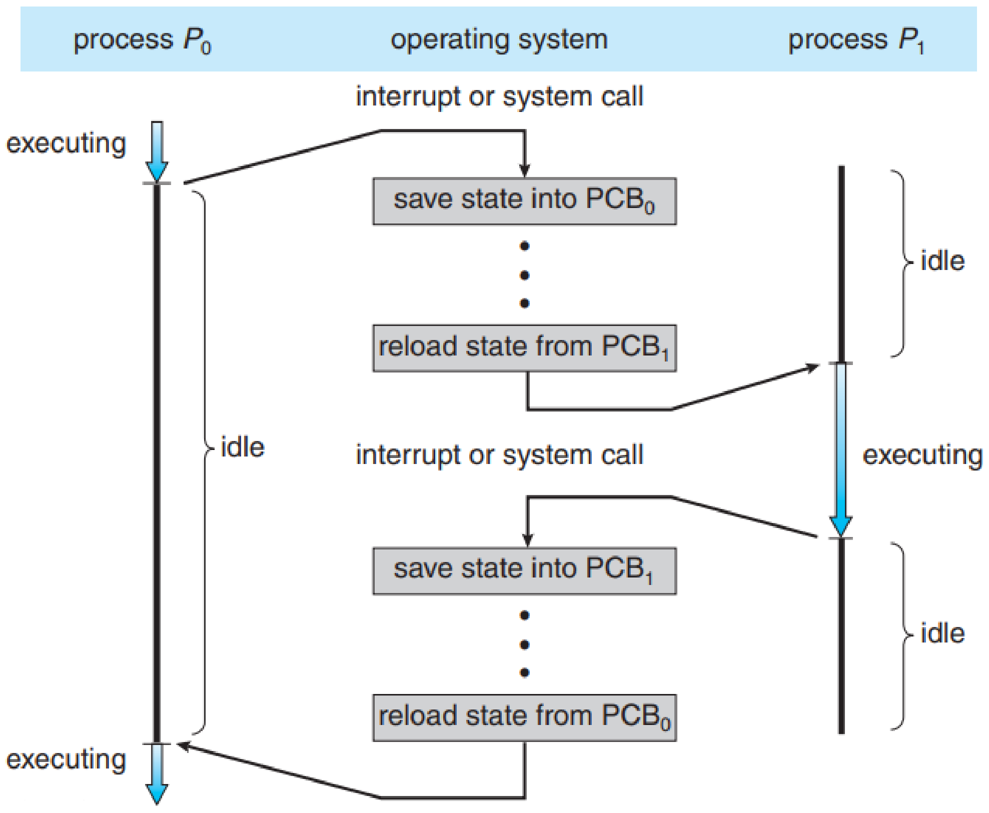


Diagram showing CPU switch from process to process.

SCHEDULER

There are two types of schedulers in operating system. These are Long Term Scheduler and Short-Term Scheduler.

1.**Long Term Scheduler**: Long term scheduler is also known as job scheduler. It chooses the processes from the pool (secondary memory) and keeps them in the ready queue maintained in the primary memory.

Long term Scheduler is necessary to ensure a degree of multiprogramming. It is important that the long-term scheduler make a careful selection of both I/O and CPU- bound processes. I/O-bound tasks are which use much of their time in input and output operations while CPU-bound processes are which spend their time on the CPU. The job scheduler increases efficiency by maintaining a balance between the two.

2.**Short-Term Scheduler**: Short Term Scheduler is also known as CPU-Scheduler. It selects the process from the ready state to the running state. Short-term scheduler only selects the process to schedule, it does not load the process on running. The Job of the short-term scheduler can be very critical in the sense that if it selects job whose CPU burst time is very high then all the jobs after that, will have to wait in the ready queue for a very long time. **The dispatcher** is responsible for loading the process selected by the Short-term scheduler on the CPU.

Job of the dispatcher:

1. Switching context.
2. Switching to user mode.
3. Jumping to the proper location in the newly loaded program.

3.**Medium Term Scheduler**: Medium term scheduler takes care of the swapped-out processes. If the running state processes needs some IO time for the completion, then there is a need to change its state from running to waiting.

Medium term scheduler is used for this purpose. It removes the process from the running state to make room for the other processes. Such processes are the swapped-out processes and this procedure is called swapping. The medium-term scheduler is responsible for suspending and resuming the processes.

It reduces the degree of multiprogramming. The swapping is necessary to have a perfect mix of processes in the ready queue.

There are various algorithms which are used by the Operating System to schedule the processes on the processor in an efficient way.

The Purpose of a Scheduling algorithm

1. Maximum CPU utilization
2. Fare allocation of CPU
3. Maximum throughput
4. Minimum turnaround time
5. Minimum waiting time
6. Minimum response time

There are some algorithms used in scheduling problem such as:

1. **First Come First Serve**
   It is the simplest algorithm to implement. The process with the minimal arrival time will get the CPU first. The lesser the arrival time, the sooner will the process gets the CPU. It is the non-pre-emptive type of scheduling.
2. **Round Robin**
   In the Round Robin scheduling algorithm, the OS defines a time quantum (slice). All the processes will get executed in the cyclic way. Each of the process will get the CPU for a small amount of time (called time quantum) and then get back to the ready queue to wait for its next turn. It is a pre-emptive type of scheduling.
3. **Shortest Job First**
   The job with the shortest burst time will get the CPU first. The lesser the burst time, the sooner will the process get the CPU. It is the non-pre-emptive type of scheduling.
4. **Shortest remaining time first**
   It is the pre-emptive form of SJF. In this algorithm, the OS schedules the Job according to the remaining time of the execution.

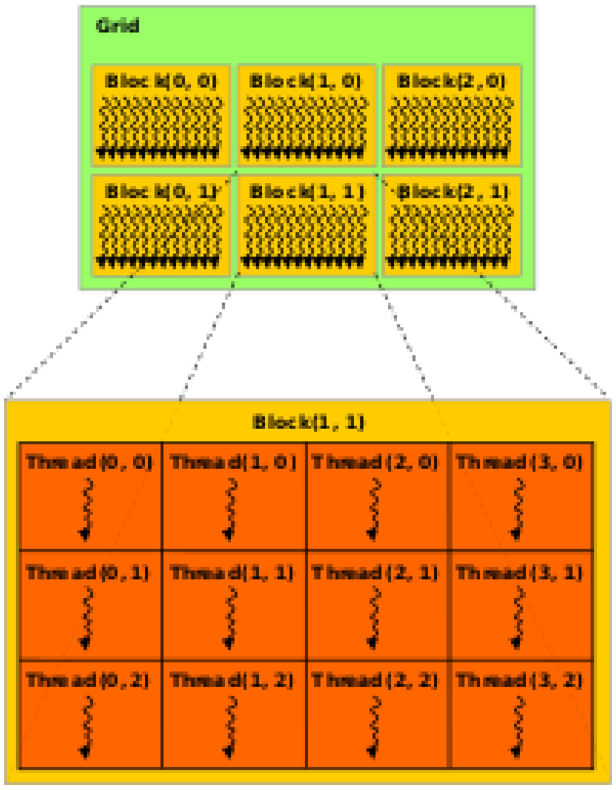
5. Priority based scheduling

In this algorithm, the priority will be assigned to each of the processes. The higher the priority, the sooner will the process get the CPU. If the priority of the two processes is same then they will be scheduled according to their arrival time.

6. Highest Response Ratio Next

In this scheduling Algorithm, the process with highest response ratio will be scheduled next. This reduces the starvation in the system.

Thread-Block



A **thread block** is a programming abstraction that represents a group of threads that can be executed serially or in parallel. For better process and data mapping, threads are grouped into thread blocks.

Multiple blocks are combined to form a grid. All the blocks in the same grid contain the same number of threads. The number of threads in a block is limited, but grids can be used for computations that require a large number of thread blocks to operate in parallel and to use all available multiprocessors.

As many parallel applications involve multidimensional data, it is convenient to organize thread blocks into 1D, 2D or 3D arrays of threads. The blocks in a grid must be able to be executed independently, as communication or cooperation between blocks in a grid is not possible.

1Dimensional-blocks

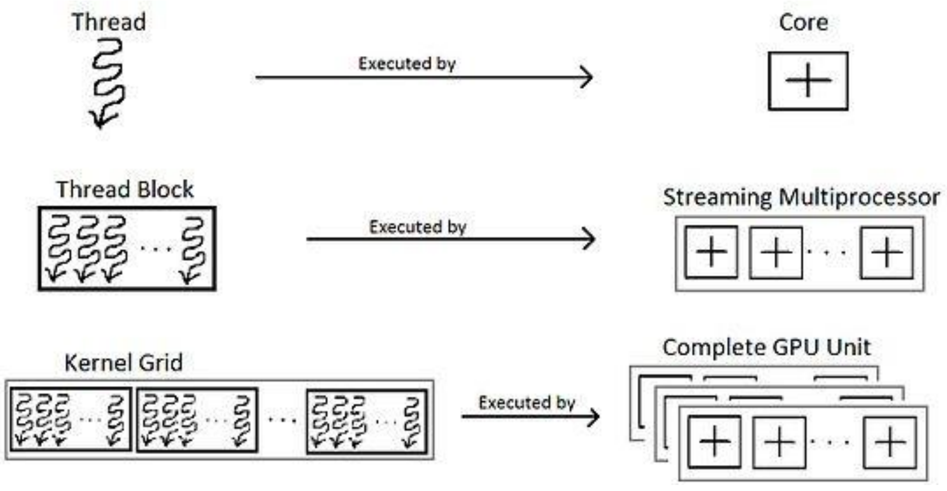
Every thread in CUDA is associated with a particular index so that it can calculate and access memory locations in an array.

Ex: Consider an example in which there is an array of 512 elements. One of the organization structure is taking a grid with a single block that has a 512 threads. Consider that there is an array C of 512 elements that is made of element wise multiplication of two arrays A and B which are both 512 elements each. Every thread has an index i and it performs the multiplication of i^{th} element of A and B and then store the result in the i^{th} element of C. i is calculated by using blockIdx (which is 0 in this case as there is only one block), blockDim (512 in this case as the block has 512 elements) and threadIdx that varies from 0 to 511 for each block.

The thread index i is calculated by the following formula:

i = blockIdx.x*blockDim.x+threadIdx.x

Here the value of i lies between 0-511 both inclusive. Now we can parallelly compute the multiplication operation.



2D Indexing

In the same way in particularly complex grids, the blockId as well as the threadId need to be calculated by each thread depending on geometry of the grid. Consider, a 2-dimensional Grid with 2-dimensional blocks. The threadId and the blockId will be calculated by the following formulae:

$$blockId = blockIdx.x + blockIdx.y * gridDim.x;$$
$$threadId = blockId * (blockDim.x * blockDim.y) + (threadIdx.y * blockDim.x) + threadIdx.x$$

GPU-MEMORY-HIERARCHY

SRAM:

Static random-access memory (static RAM or SRAM) is a type of random-access memory (RAM) that uses latching circuitry (flip-flop) to store each bit. SRAM is volatile memory; data is lost when power is removed.

SRAM will hold its data permanently in the presence of power, while data in DRAM decays in seconds and thus must be periodically refreshed.

There are two key features to SRAM - Static Random Access Memory, and these set it out against other types of memory that are available:

- **The data is held statically:** This means that the data is held in the semiconductor memory without the need to be refreshed as long as the power is applied to the memory.
- **SRAM memory is a form of random-access memory:** A random access memory is one in which the locations in the semiconductor memory can be written to or read from in any order, regardless of the last memory location that was accessed.

Access to the SRAM memory cell is enabled by the Word Line. This controls the two access control transistors which control whether the cell should be connected to the bit lines. These two lines are used to transfer data for both read and write operations.

DRAM: As the name DRAM, or dynamic random access memory, implies, this form of memory technology is a type of random access memory. It stores each bit of data on a small capacitor within the memory cell. The capacitor can be either charged or discharged and this provides the two states, "1" or "0" for the cell.

Since the charge within the capacitor leaks, it is necessary to refresh each memory cell periodically. This refresh requirement gives rise to the term dynamic - static memories do not have a need to be refreshed.

The need to refresh DRAM demands more complicated circuitry and timing than SRAM. This is offset by the structural simplicity of DRAM memory cells: only one transistor and a capacitor are required per bit, compared to four or six transistors in SRAM. This allows DRAM to reach very high densities with a simultaneous reduction in cost per bit. Refreshing the data consumes power and a variety of techniques are used to manage the overall power consumption.