Module-2

Process Management

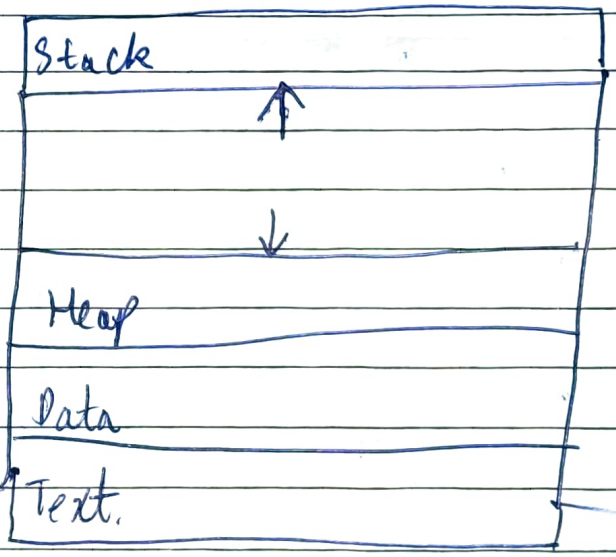
Hi

Page No.
Date

Unit 2.1 Processes

- A process is a program in execution. It includes the program code, data and its execution context such as registers and program counter.
- When a program is loaded into the memory and it becomes a process it can be divided into four section stack, heap, text & data.

Simple Layout of ~~mem~~ process in main memory:-



Stack:- Process stack contain temporary data such as method/fn parameters, return address and local variable.

Heap:- It is dynamically allocated memory to a process during its run time.

Text:- It includes the current activity represented by value of program counter & registers

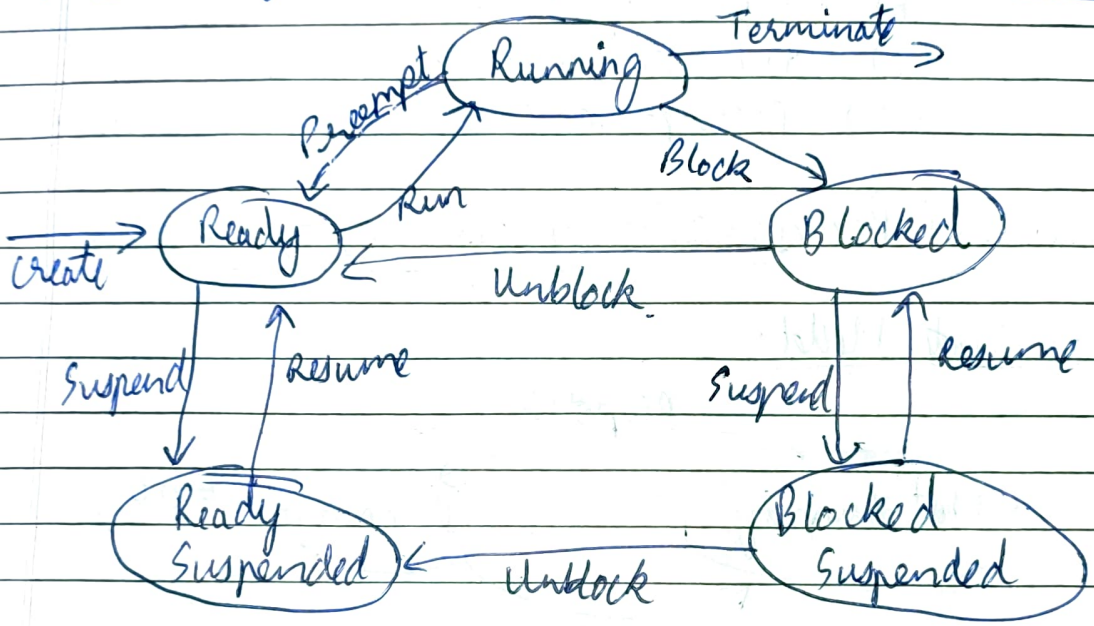
Data:- It contains global and static variables.

Page No.
Date

Program

A program is a piece of code which may be single line or millions of line. It is basically a collection of instructions that performs a specific task when executed by a computer.

2) States of Process (Process Creation, Suspension & termination)



i) **New**: Newly created process (or) being created process.

ii) **Ready**: After creation process moves to ready state, i.e the process is ready for execution.

iii) **Run**: Currently running process in CPU (only one process at a time)

iv) **Wait (or Block)**: When process requests i/o access.

v) **Complete (Terminated)**: The process is executed.

vi) **Suspended Ready**: When ready queue is full some process is moved to suspended ready state.

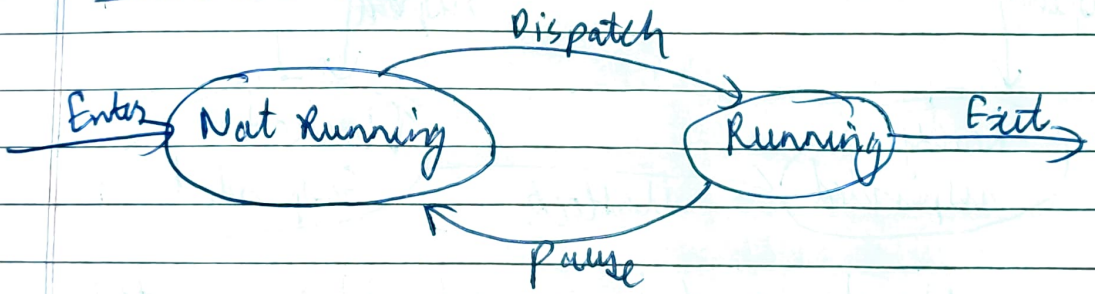
vii) **Suspended block**: When waiting queue becomes full.

Context Switching: Process of saving the context of one process and loading context of another process is known as Context switching.
(loading & unloading the process from running state to ready state)

Reason: (a) When high priority process comes to ready state
(b) An interrupt occurs
(c) User & kernel-mode switch
(d) Preemptive CPU scheduling is used.

(3) Different Process states models (2, 3, 5 & 7)

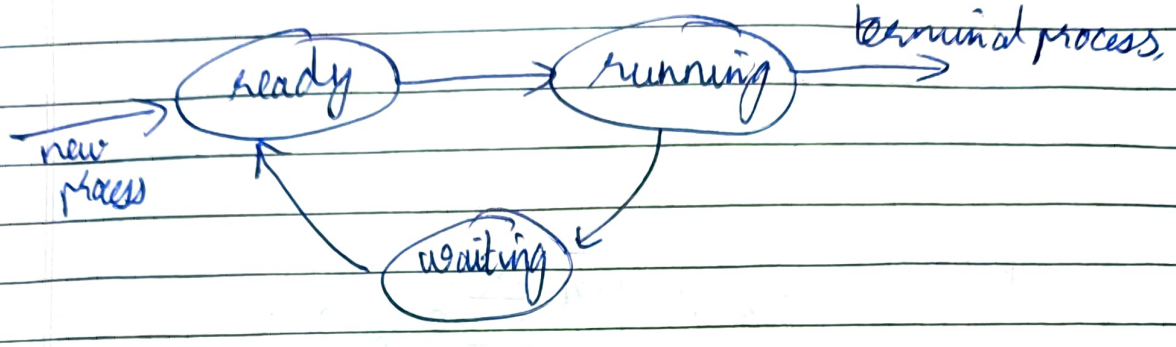
(a) Two-State Model:



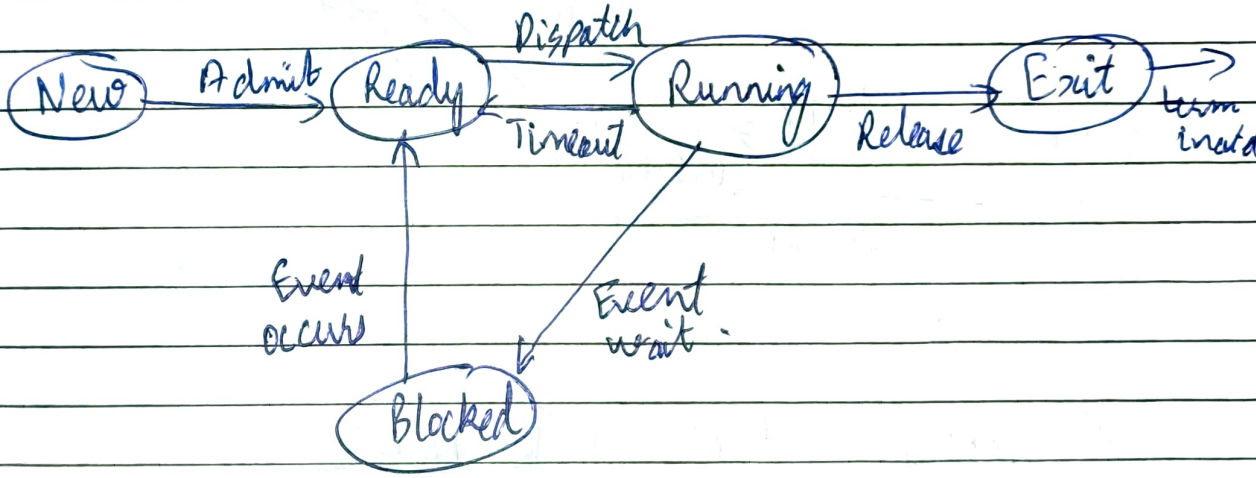
- It is the simplest model in process state.
- Firstly OS creates a new process, it also creates a process control block for the process so that the process can enter into system in a non-running state. If any process exit/leaves the system, then it is known to OS.
- When process is in running state & has exhausted its time slice or voluntarily yields the CPU (by making I/O request) it moves to non-running state & back to queue.
- When the event or resource the process was waiting for

becomes available it transitions back to running state and continues its execution.

(b) Three-state Model:



(c) Five-state Process Model:

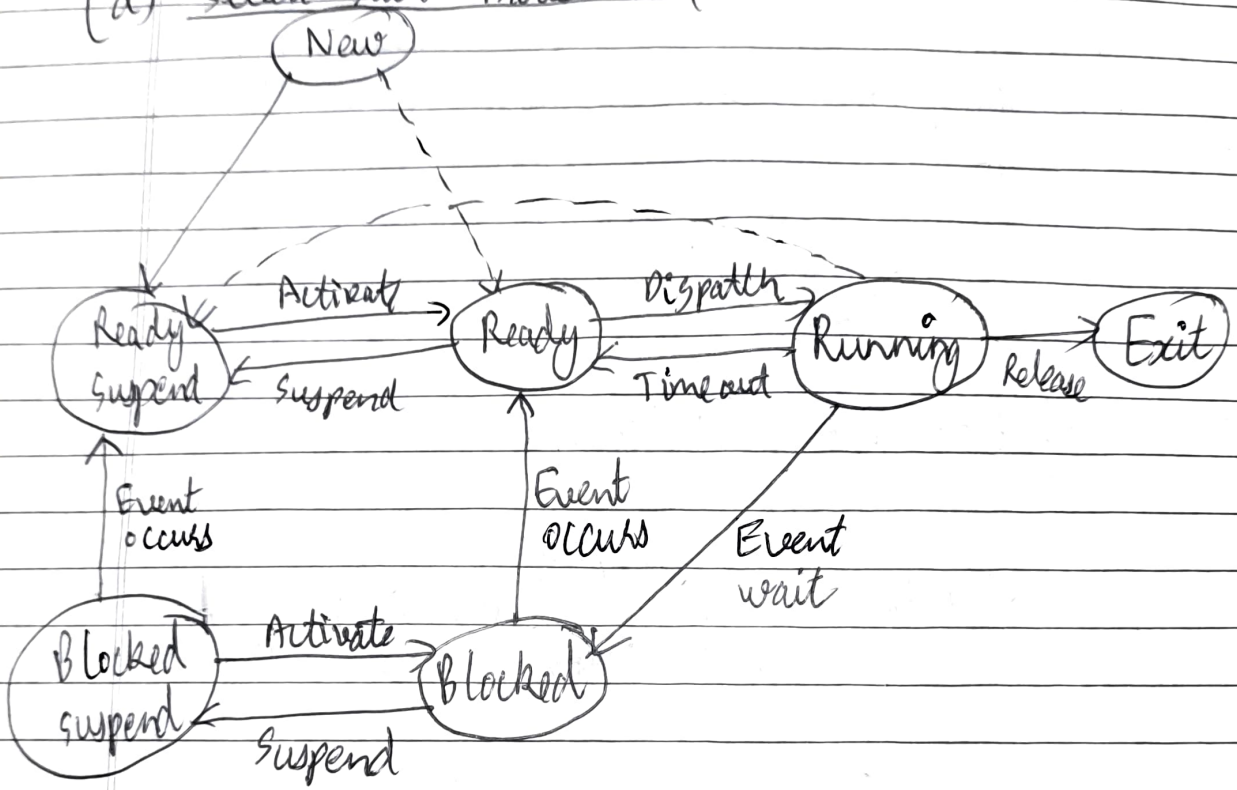


- Model consists of five states. The model works when new job/process occurs in queue; it first gets admitted in queue after that it goes in the ready state.
- Now in ready state it goes in running state. In running state a process has 2 conditions either the process goes to the event wait or the process gets timeout.
- If process has timeout it goes to ready state as process has not completed its execution.
- If process has an event wait condition then process

goes to blocked state and after that to Ready state.

- If both conditions are true then process goes to running state after dispatching after which process gets released and at last it is terminated.

(d) Seven-state Process model :



(4) Process Control Block (PCB)

Process ID
~~Process state~~ (security)
Process State
Priority
General Purpose registers
List of Open files
List of Open devices

Process ID : When process is created a unique ID is assigned to process which is used for unique identification of process in system.

Program Counter : It stores the counter which contains the address of next instruction that is to be executed for process.

Process State: It stores respective state of process.

Priority: Every process has its own priority. The process with highest priority among processes get the CPU first.

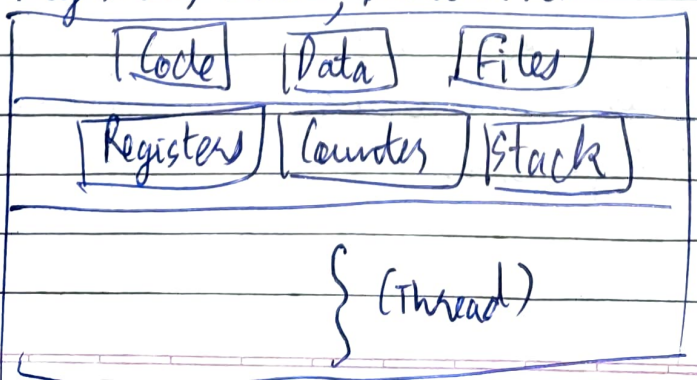
Registers: Every process has its own registers which are used to hold the data which is generated during execution of Process.

List of open files: During Exec every process uses same files which needs to be present in main memory.

List of open devices: It also maintain list of all open devices which are used during process execution.

Unit 2.2 Threads

- Threads is a sequential flow of tasks within process.
- Threads are used to increase the performance of applications.
- Threads are also termed as lightweight processes as they share common resources.
- Each thread has its own program counter, stacks, registers, code, data & files.

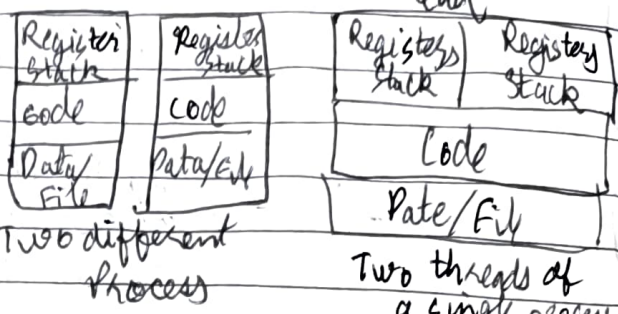


Single threaded process

Components of Thread

1) Program Counter

Importance of Threads:



Two different Process

Two threads of a single process.

- Since threads use the same data and code, the operational cost between threads is low.
- Creating and terminating a thread is faster compared to creating or terminating process.
- Context switching is faster in threads as compared to process.

5) Multithreading

- The idea is to divide a single process into multiple threads instead of creating a whole new process.
- It is done to achieve parallelism and to improve the performance of application as it is faster.

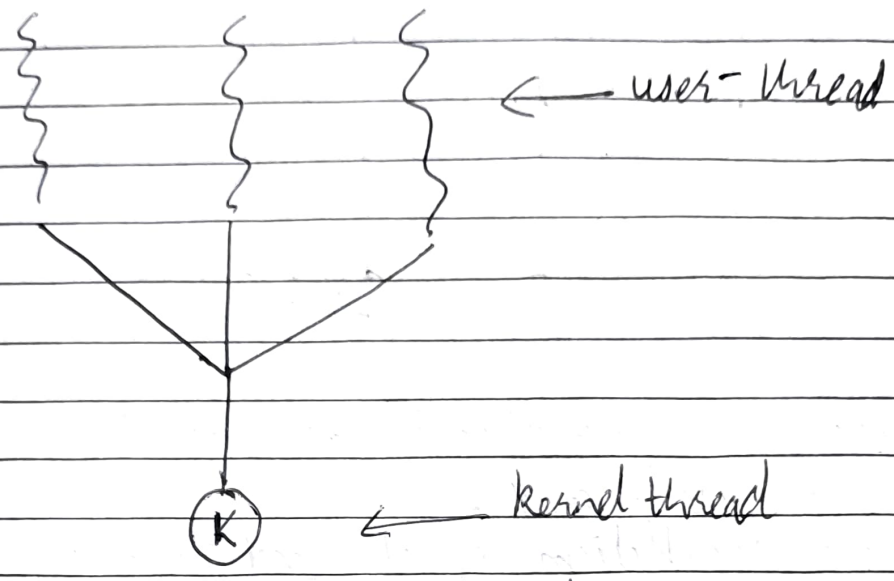
(a) Resource sharing: Threads of single process share the same resources such as code & data/fil

(b) Responsiveness. Program responsiveness enables program to run even if part of program is blocked.

(c) Economy: It's more economical to use threads as they share resource while processes are expensive.

6) Multithreading Models

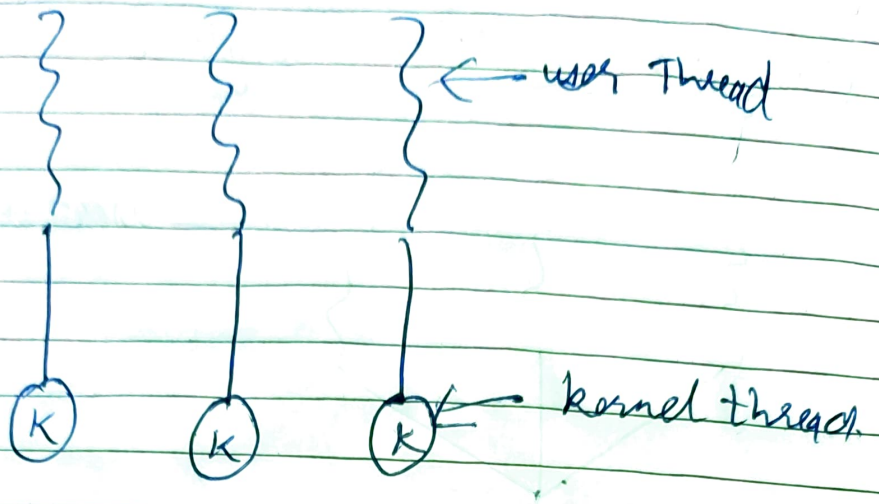
(a) Many to one model: (Single Processor) {user-level thread}.



- In this model multi user-level threads are managed by single ~~thread~~ kernel-thread or process
- The O.S is unaware of user-level threads and schedules entire process as a single entity.
- User-level threads can be created, scheduled and management entirely by application without kernel involvement.
- It is simple and lightweight but it lacks true parallelism bcoz if one thread in process blocks, it can block all threads in process.
- It will be slower than other models as only single Kernel is involved.

Eg: Web server implemented using many-to-one threading. In this scenario a web server uses many to one threading model to handle incoming requests. Each client is represented by user-level thread.

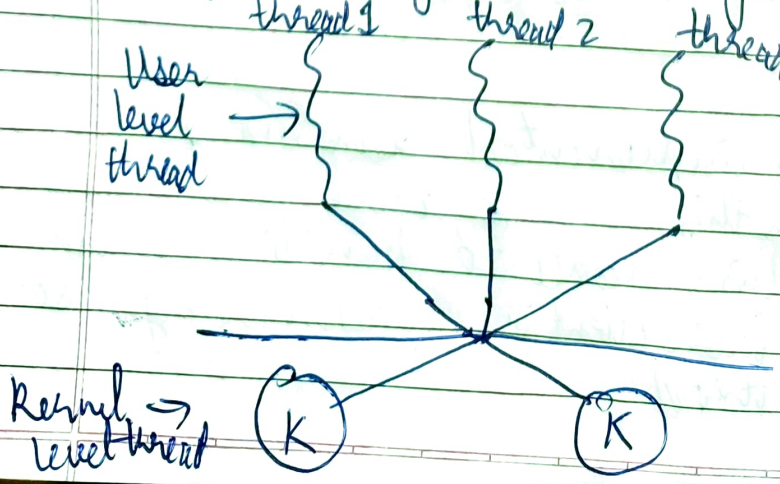
(b) One-to-one model (Kernel level thread)



- Here each user thread corresponds to separate Kernel level thread.
- Here parallelism is achieved as threads can run concurrently on multiple CPU cores.
- It can be relatively heavy in terms of system resource usage because each thread requires its own Kernel data structure and expensive.
- Its performance is not good in comparison to many-to-one.

Eg: Windows Series & Linux Operating System use this model.

(c) Many-to-Many Model: (Hybrid thread)



- This is a model which is combination of many-to-one and one-to-one.
- The operating system is aware of both user-level and kernel-level threads and can perform scheduling at both levels.
- This model attempts to provide a balance between lightweight user-level and true parallelism offered (or multitasking) offered by kernel-level threads.
- It allows scalability where number of threads can grow and shrink based on system load and application needs. It provides flexibility to adapt to varying workloads.

Eg: Operating systems and runtime environments like Java's JVM often employ a many-to-many threading model.

(7) Thread Implementations

(a) User-level Thread:

- The user level thread is ignored by O.S. User threads are simple to implement and are done by the user.
- The entire process is blocked if a user executes a user-level operation of thread blocking.
- User level threads are managed as single threaded processes by kernal level thread.

Pros:

a) Easier to implement
b) More Effective & efficient.

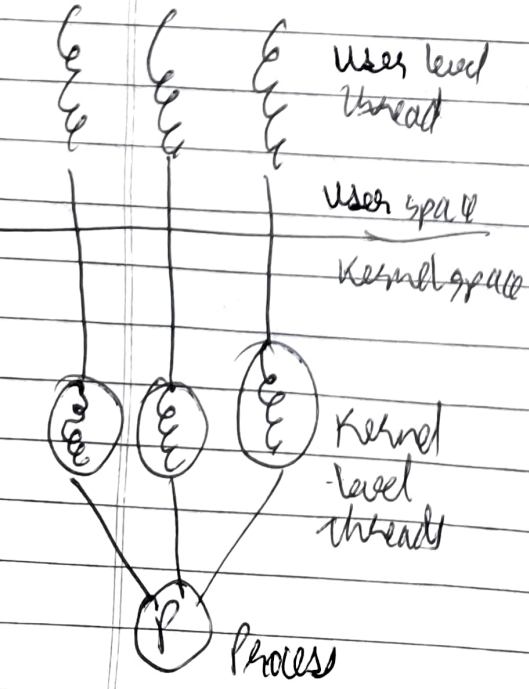
c) Context switching takes less time.
d) Threads may be easily be created, switched & synchronised without need of process interaction.

Cons:

a) Threads at user level are not coordinated with the kernel.
b) Entire operation is halted if a thread creates page fault.

(b) Kernel-level Thread:

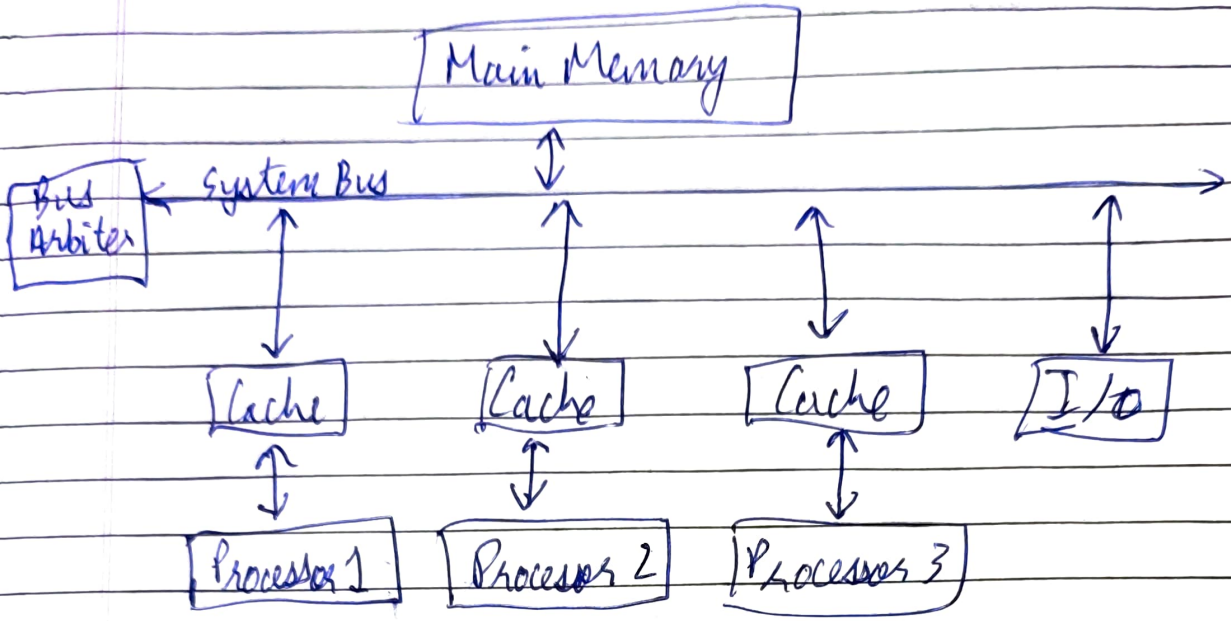
- The operating is recognised by kernel thread.
- Each thread and process in kernel-level thread has it own thread control block as well as process control block in the system.
- The kernel is aware of all threads and controls them.
- Kernel threads are more complex to build.
- Its context switch time is longer.



Pros: a) All threads are kn
completely aware by

Cons: a) All threads are managed by Kernel thre
b) Kernel threads are more complex.
c) Kernel threads are slower than user-lev threads.

(4) Symmetric Multiprocessing



- SMP (Symmetric multiprocessing) is computer processing done by multiple processors that share common operating system (OS) and memory.
- In Symmetric multiprocessing, the processors share the same input/output (I/O) bus or data path.
- SMP systems can dynamically balance workload among computers to serve more users faster.

Eg: SMP are used primarily in resource-intensive computing environments. These are environment that require large amount of computing power to execute applications such as online transactions processing where user access the same database in relatively simple set of instructions.

Advantages:

a) Increased productivity or output as multiple processors decrease time for task.
b) Cost effective
c) Reliability: If processor fails, the whole system does not fail.
d) Performance: