Process Management
CPU Scheduling

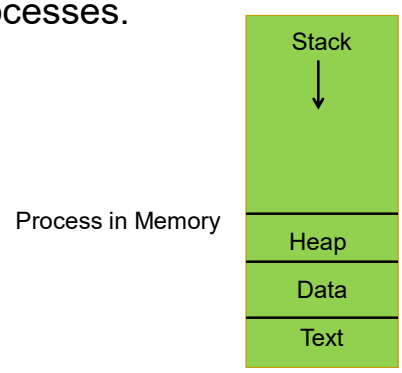
Introduction

- A process is a part of program in execution having a definite state.
- It includes the current activity represented by the current value of the program counter and the contents of the processor's register.
- Occasionally it can include a process stack containing temporary data and a data section containing global variables
- It may also include a heap; memory that is dynamically allocated during process run time.
- A program by itself is not a process; a program is a passive entity, such as an executable file containing list of instructions and stored as an executable file, a process is an active entity with a program counter and allotted resources.
- A program becomes a process when an executable file is loaded into the memory.

2

Introduction

- For e.g. a user invoking multiple instance of a web bowser; each instance is a separate process, although program is same.
- A process, while executing, may also spawn (fork) new processes.



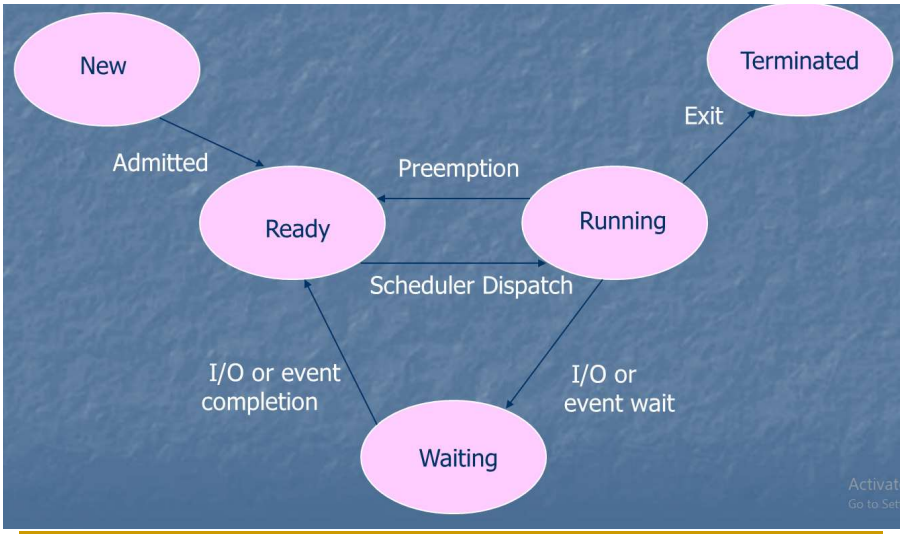
3

Process States

- As a process executes, it changes state. Each process may be in one of the following states :
  - New : the process is being created
  - Running : instructions are being executed
  - Waiting : the process is waiting for some event to occur
  - Ready : the process is waiting to be assigned to a processor
  - Terminated: the process has finished execution

4

Process State Transition Diagram



5

Process Control Block

- A Process Control Block (PCB) is associated with each process that serves as a repository for process information and helps the system in tracking progress of each process
  - Process state
  - Program counter
  - CPU registers: depending on the CPU architecture, it may include accumulator, index register, stack pointer, General purpose registers.
  - CPU- scheduling information: priority, pointers to scheduling queue
  - Memory management information: depending on the memory system, it may include the value of the base and limit register, the page table or the segment table.
  - Accounting information : pid, % resource utilization
  - I/O status information: list of I/O devices allocated, list of open files

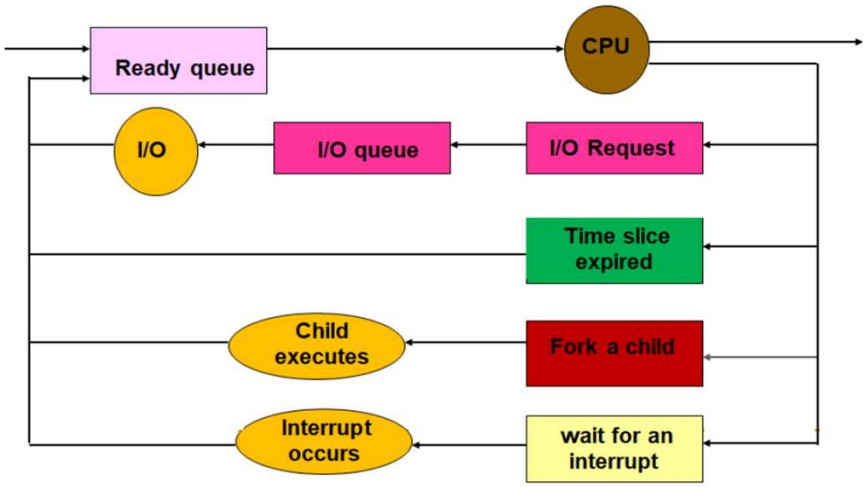
6

Process Scheduling

- In a multiprogramming environment, the objective of the system is to ensure some process is running at all the time, to maximize CPU utilization.
- The CPU scheduler, based on the underlying criteria, needs to select one process and allocate it the CPU.
- In order to efficiently manage all the processes, the system maintains different queues, as per the current state of the processes.

7

Scheduling Queues



8

Schedulers

- The system picks various processes from different queues for the scheduling purpose based on certain criteria
- This selection is done by a module known as the Scheduler
  - Long-term scheduler
  - Short-term scheduler
  - Medium-term scheduler

9

Long-term Scheduler

- This scheduler, when present, admits low priority jobs that may be used as fillers to keep the system resources busy during low activity periods of interactive programs
- Primary objective is to provide a balanced mix of processor bound and I/O bound jobs to the short-term scheduler; keeps resource utilization at the desired level
- Frequency of invocation is lower as compare to other two schedulers
- In terms of PST diagram, this scheduler is in charge of New to Ready state transitions
- Normally not found in systems without support for a batch

10

Medium-term Scheduler

- After executing for a while, a running process may become suspended, waiting for the completion of an external event; such processes cannot make any progress towards completion until the event is over.
- In order to maintain the supply of ready processes to the short term scheduler, it is beneficial to remove such suspended processes from the memory
- Saving the image of a suspended process in secondary storage is called as **Swapping** and the process is said to be Swapped out or Rolled out

11

Medium-term Scheduler

- This scheduler is in charge of handling the swapped out processes
- When the suspension condition is removed, it attempts to allocate the required amount of memory to the swapped out process and make it ready
- In terms of the PST diagram, this scheduler is in charge of Waiting to the Ready state transitions
- Invoked whenever memory is vacated by a departing process or when the supply of ready processes falls below a certain limit
- Needed only when swapping is used by the underlying OS

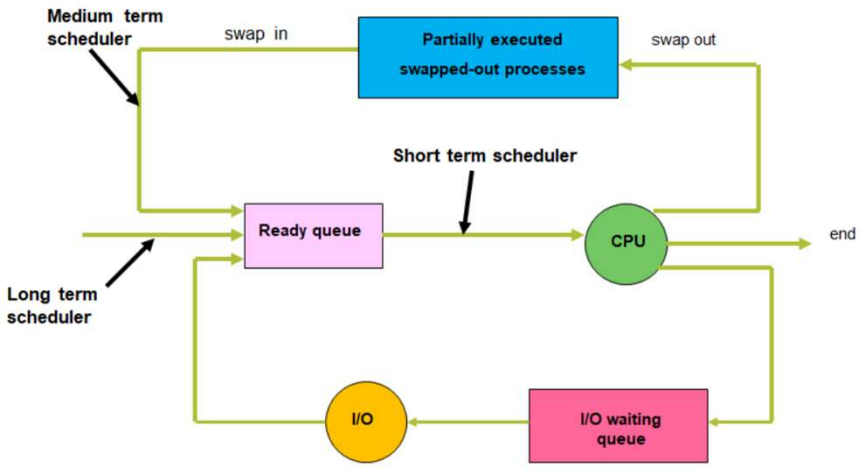
12

Short-term Scheduler

- Allocates the processor/CPU among the pool of ready processes resident in memory
- In charge of Ready to Running state transitions
- Invoked for each process switch, i.e. whenever an event causes the global state of the system to change
- Systems that support more than one type of scheduler must provide for proper support for communication and interaction among the schedulers for attaining satisfactory and balanced performance

13

Types of Schedulers



14

Scheduling Criteria

- To choose the appropriate scheduling algorithm for a system, the system programmer must consider various properties of the algorithms. The criteria include :
- **CPU utilization**: The average fraction of time during which the processor is busy executing user program or the OS
- **Throughput**: The amount of work completed in a unit time
- **Turnaround time**: The time that elapses from the moment a program or a job is submitted until it is completed by the system. It is the sum of time spent waiting to get into memory, waiting in the ready queue, executing on the CPU and doing I/O

15

Scheduling Criteria

- **Waiting time**: Time that a process or job spends waiting for the resource allocation due to the contentions with others; it is the penalty imposed for sharing resources with others. It is the sum of the periods spent waiting in the ready queue
- **Response time**: Time that elapses from the moment the last character of a command line launching program or a transaction is entered until the first result appears on the screen; more prevalent in an interactive system
- It is desirable to have maximum CPU utilization and throughput and minimum waiting time, turnaround time and the response time

16

First Come First Served Scheduling

- Simplest scheduling algorithm; the process that requests the CPU first is allocated the CPU first
- Implementation is easily managed by a FIFO queue; when the process enters the ready queue, its PCB is linked onto the tail of the queue. When the CPU is free, it is allocated to the process at the head of the queue. The running process is then removed form the queue
- It's a non preemptive algorithm
- Average waiting time is long

17

FCFS Example

- Let three processes P1, P2 and P3 all arrive at time t_0

Process	CPU burst time
P1	24 ms
P2	3 ms
P3	3 ms

What is the average waiting time and Turnaround time, if they are served in the order P1, P2, P3 ?

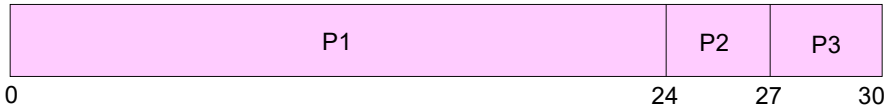
What are the times if they are served in the order P2, P3, P1 ?

18

FCFS Example

- Gantt chart

Arrival order : P1, P2, P3



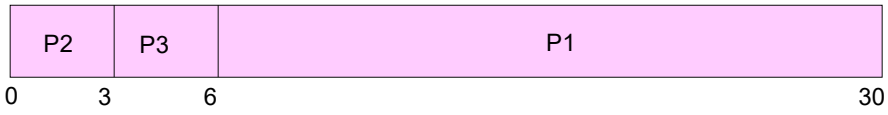
Process	Waiting time	Turnaround time
P1	0 ms	24 ms
P2	24 ms	27 ms
P3	27 ms	30 ms

Average waiting time: 17 ms
Average turnaround time 27 ms

19

FCFS Example

Arrival order : P2, P3, P1



Process	Waiting time	Turnaround time
P2	0 ms	3 ms
P3	3 ms	6 ms
P1	6 ms	30 ms

Average waiting time 3 ms
Average turnaround time 13 ms

20

Shortest Job First Scheduling

- Takes into consideration the length of next CPU burst of each process
- When CPU is idle it is assigned to the process having the smallest next CPU burst; two processes having the same burst time are processed in FCFS manner
- Gives the minimal average waiting time for a given set of processes
- Algorithm requires knowledge of the length of next CPU burst of each process
- Frequently used in long–term scheduling; user can specify the process time limit while submitting the job to the batch system

21

Shortest Job First Scheduling

- Not implemented at the short term scheduling level as there is no way to know the length of the next CPU burst
- However SJF can be approximated by predicting the next CPU burst as an exponential average of the measured lengths of previous CPU bursts
- Let t_n be the length of the n^{th} CPU burst (most recent information) and ζ_{n+1} be the predicted value for the next CPU burst, ζ_n stores the past history. Then for $0 \leq \lambda \leq 1$

$$\zeta_{n+1} = \lambda\, t_n + (1 - \lambda)\, \zeta_n$$

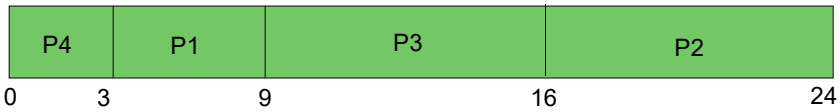
The parameter λ controls the relative weight of recent and past history . If $\lambda = 0$, then $\zeta_{n+1} = \zeta_n$ and recent history has no effect; if $\lambda = 1$, then $\zeta_{n+1} = t_n$, and only the most recent CPU burst matters. More commonly , $\lambda = ½$, so recent and past history are equally weighted

22

Shortest Job First Scheduling

- Let P1, P2, P3 and P4 all arrive at time t_0

Process	Burst time
P1	6 ms
P2	8 ms
P3	7 ms
P4	3 ms



Process	Waiting time	Turnaround time
P4	0 ms	3 ms
P1	3 ms	9 ms
P3	9 ms	16 ms
P2	16 ms	24 ms

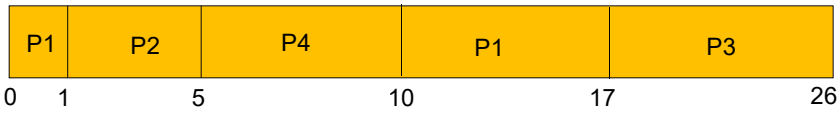
Average waiting time : 7 ms Average turnaround time 13 ms

23

Shortest Remaining Time Next Scheduling

- Preemptive version of SJF scheduling

Process	Arrival time	Burst time
P1	0	8 ms
P2	1	4 ms
P3	2	9 ms
P4	3	5 ms

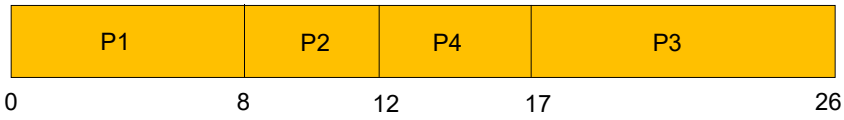


Process	Waiting time	Turnaround time
P1	9 ms	17 ms
P2	0 ms	4 ms
P4	2 ms	7 ms
P3	15 ms	24 ms

24

Using SJF

Process	Arrival time	Burst time
P1	0	8 ms
P2	1	4 ms
P3	2	9 ms
P4	3	5 ms



Process	Waiting time	Turnaround time
P1	0 ms	8 ms
P2	7 ms	11 ms
P4	9 ms	14 ms
P3	15 ms	24 ms

The average waiting time of SRTN is 6.5 ms while with SJF it is 7.75 ms
The average turnaround time of SRTN is 13 ms while with SJF it is 14.25 ms

25

- SRTN scheduling

Process	Arrival time	Burst time
P1	0	8 ms
P2	1	4 ms
P3	2	9 ms
P4	3	5 ms

Assume that the context switching time is 0.2 ms

Process	Waiting time	Turnaround time
P1	9.6 ms	17.6 ms
P2	0.2 ms	4.2 ms
P4	2.4 ms	7.4 ms
P3	15.8 ms	24.8 ms

26

Priority Based Scheduling

- Each process is assigned a priority, the higher priority processes are evaluated first
- Priorities can be defined internally (memory requirements, time limits, no. of open files, ratio of average I/O bursts to CPU bursts) or externally (type, importance, political)
- Can be preemptive or non preemptive
- A major problem is of indefinite blocking or starvation; can be solved by aging

27

Non Preemptive Priority Based Scheduling

- Let P1, P2, P3, P4, P5 all arrive at time t_0

Process	Burst time	Priority
P1	10 ms	3
P2	1 ms	1
P3	2 ms	4
P4	1 ms	5
P5	5 ms	2

Process	Waiting time	Turnaround time
P2	0 ms	1 ms
P5	1 ms	6 ms
P1	6 ms	16 ms
P3	16 ms	18 ms
P4	18 ms	19 ms

The average waiting time is 8.2 ms and average Turnaround time is 12 ms

28

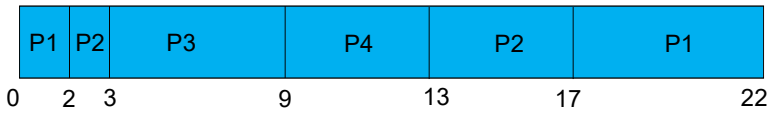
Preemptive Priority Based Scheduling

- Consider the arrival time, burst time and priority of four processes as shown below. Find the average waiting time and average turnaround time using

Process	Arrival time	Burst time	Priority
P1	0	7 ms	4
P2	2	5 ms	3
P3	3	6 ms	1
P4	5	4 ms	2

29

Process	Arrival time	Burst time	Priority
P1	0	7 ms	4
P2	2	5 ms	3
P3	3	6 ms	1
P4	5	4 ms	2



Process	Waiting time	Turnaround time
P1	15 ms	22 ms
P2	10 ms	15 ms
P3	0 ms	6 ms
P4	4 ms	8 ms

The average waiting time is 7.25 ms
and average Turnaround time is 12.75 ms

30

Round- Robin Scheduling

- Designed specifically for time sharing systems
- Can be also called as FCFS with preemption
- Each process gets a fixed time quantum or time slice; no process gets more than 1 time slice in a row(provided there are other contenders)
- The system maintains the ready queue as the FIFO queue, new processes are added at the end, scheduler picks the processes from the front, processes requiring more than one time slice are preempted and added at the end of the queue
- Performance is heavily dependent on the size of the time slice, larger time slice reduce the algorithm to FCFS and smaller slice increases the context switching time

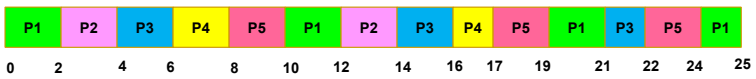
31

Round- Robin Scheduling

Consider 5 processes all arriving at t_0 with their burst time as given:

Find the average waiting time and turnaround time with time slice as 2ms

Process	Burst time			
P1	7 ms	5	3	1
P2	4 ms	2	0	0
P3	5 ms	3	1	0
P4	3 ms	1	0	0
P5	6 ms	4	2	0



Process	Waiting Time	Turnaround Time
P1	18 ms	25 ms
P2	10 ms	14 ms
P3	17 ms	22 ms
P4	14 ms	17 ms
P5	18 ms	24 ms

32

Round- Robin Scheduling

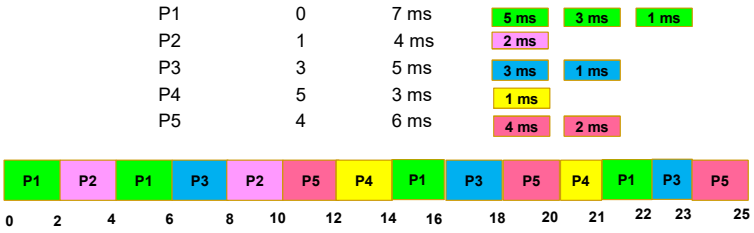
- Consider 5 processes with their arrival time and burst time as given below. Find the average waiting time and turnaround time with time slice as 2 ms

Process	Arrival time	Burst time
P1	0	7 ms
P2	1	4 ms
P3	3	5 ms
P4	5	3 ms
P5	4	6 ms

33

Round- Robin Scheduling

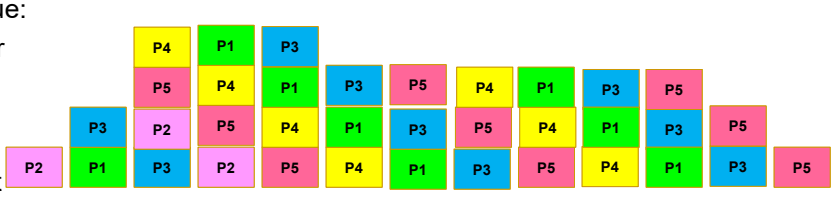
Process	Arrival time	Burst time
P1	0	7 ms
P2	1	4 ms
P3	3	5 ms
P4	5	3 ms
P5	4	6 ms



Queue:

Rear

Front



34

Process	Waiting Time	Turnaround Time
P1	15 ms	22 ms
P2	5 ms	9 ms
P3	15 ms	20 ms
P4	13 ms	16 ms
P5	15 ms	21 ms

Average Waiting Time is 12.6 ms
Average Turnaround Time is 17.6 ms

35

Example

- Consider the following set of processes that arrive in the order P1, P2, P3, P4, P5 all at time 0

Process	Burst time	Priority
P1	10 ms	3
P2	1 ms	1
P3	2 ms	3
P4	1 ms	4
P5	5 ms	2

- Draw Gantt charts illustrating the execution of these processes using FCFS,SJF, non preemptive priority, RR (quantum = 1) scheduling
- What is the waiting time and turnaround time of each process for each of these scheduling schemes? Also find the average waiting time and turnaround time for each scheme
- Which of the algorithm results in minimal waiting time and minimal turnaround time?

36

- FCFS
  - Average Waiting time : 9.6 ms
  - Average Turnaround time : 13.4 ms
- SJF
  - Average Waiting time : 3.2 ms
  - Average Turnaround time : 7 ms
- Priority based
  - Average Waiting time : 8.2 ms
  - Average Turnaround time : 12ms
- Round Robin
  - Average Waiting time : 5.4 ms
  - Average Turnaround time : 9.2 ms

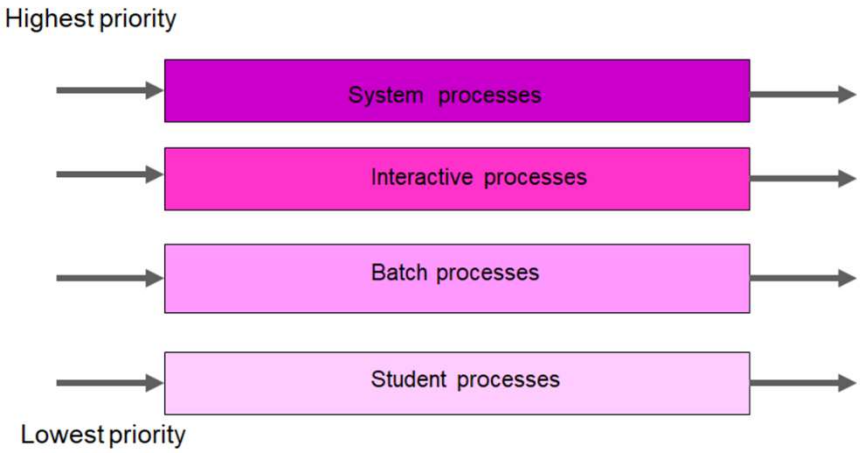
37

Multilevel Queue

- Partitions the queue into several separate queues; processes are assigned permanently to one queue based on certain property
- Each queue has its own scheduling scheme; in addition there is scheduling among the queues; each queue has absolute priority over lower priority queue
- Solaris 2 uses this scheme
- Low scheduling overhead but inflexible

38

Multilevel Queue



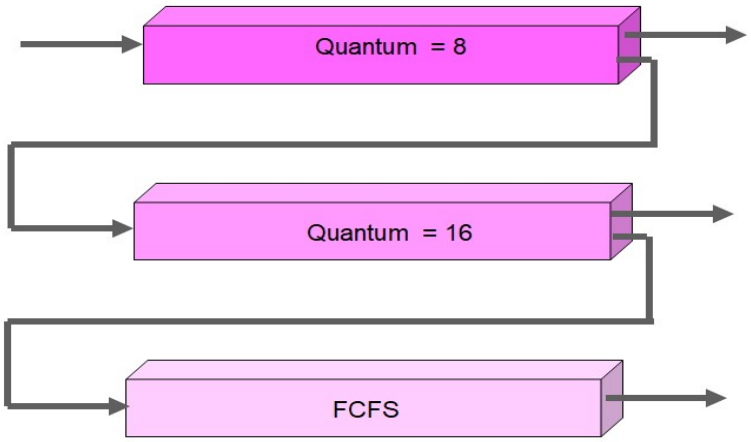
39

Multilevel Feedback Queue

- Allows processes to move between the queues; solves the problem of starvation
- The scheduler is defined by the following parameters :
  - The number of queues
  - The scheduling algorithm for each queue
  - The method used to determine when to upgrade a process to a higher priority queue
  - The method used to determine when to demote a process to a lower priority queue
  - The method used to determine which queue a process will enter when it needs service

40

Multilevel Feedback Queue



41

Multiple-Processor Scheduling

- Systems having more than one processor; can be homogeneous or heterogeneous
- Processors can share a common memory or each processor can have its own memory
- Each processor can have its own job queue (processors can lie idle) or processors can share a common queue (access needs to be synchronized)

42

Multiple-Processor Scheduling

- **Master/slave assignment**: Kernel functions always run on a particular processor. Other processors execute user processes.
  - Advantage: Resource conflict resolution simplified since single processor has control.
  - Problem: Failure of master processor? Master processor does the scheduling ==> bottleneck.
- **Peer assignment**: OS can execute on any processor. Each processor does its own scheduling from the pool of available processes. This is similar to Solaris or NT symmetric multiprocessing (SMP).

43

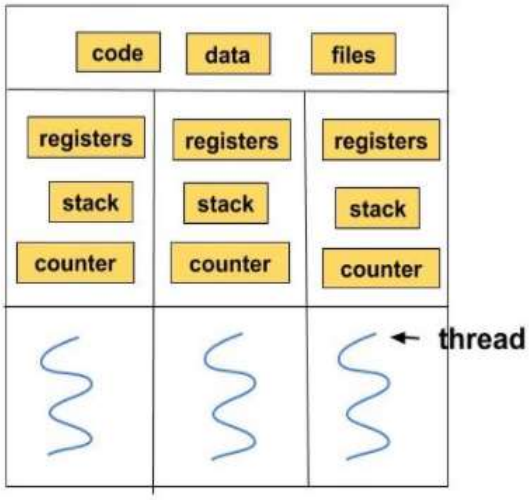
Threads

Introduction

- A thread is an execution unit that has its own program counter, a stack and a set of registers that reside in a process. Threads can't exist outside any process.
- Each thread belongs to exactly one process. The information like code segment, files, and data segment can be shared by the different threads.
- A thread is also known as lightweight process. The idea is to achieve parallelism by dividing a process into multiple threads. For example, in a browser, multiple tabs can be different threads. MS Word uses multiple threads: one thread to format the text, another thread to process inputs.

45

Introduction



46

Process Vs Thread

- The primary difference is that threads within the same process run in a shared memory space, while processes run in separate memory spaces.
- Threads are not independent of one another like processes are, and as a result threads share with other threads their code section, data section, and OS resources (like open files and signals).
- Like process, a thread has its own program counter (PC), register set, and stack space.

47

Advantages of Thread over Process

- Faster context switch: Context switch time between threads is lower compared to process context switch. Process context switching requires more overhead from the CPU.
- Effective utilization of multiprocessor system: If we have multiple threads in a single process, then we can schedule multiple threads on multiple processor. This will make process execution faster.
- Communication: Communication between multiple threads is easier, as the threads shares common address space. They do not need to use Inter-Process communication. While in process, we have to follow some specific communication technique for communication between two process.

48

Advantages of Thread over Process

- Enhanced throughput of the system: If a process is divided into multiple threads, and each thread function is considered as one job, then the number of jobs completed per unit of time is increased, thus increasing the throughput of the system.
- Responsiveness: If the process is divided into multiple threads, and one thread completes its execution, then its output can be immediately returned.
- It takes far less time to create a new thread in an existing process than to create a new process.
- It takes less time to terminate a thread than a process.

49

Types of Threads

- There are two types of threads.
  User Level Thread
  Kernel Level Thread

50

User-level Thread

- Threads in the user space designed by the application developer using a thread library to perform unique subtask.
- User threads can be easily implemented
- These threads are faster to create and manage.
- The user-level threads are managed by users and the kernel is not aware of it. The kernel manages them as if it was a single-threaded process.
- It is implemented using user-level libraries and not by system calls. So, no call to the operating system is made when a thread switches the context.
- Context switch time is shorter than the kernel-level threads.
- Each process has its own private thread table to keep the track of the threads.
- Examples: Java thread, POSIX threads, etc.

51

User-level thread

Disadvantages of User-level threads

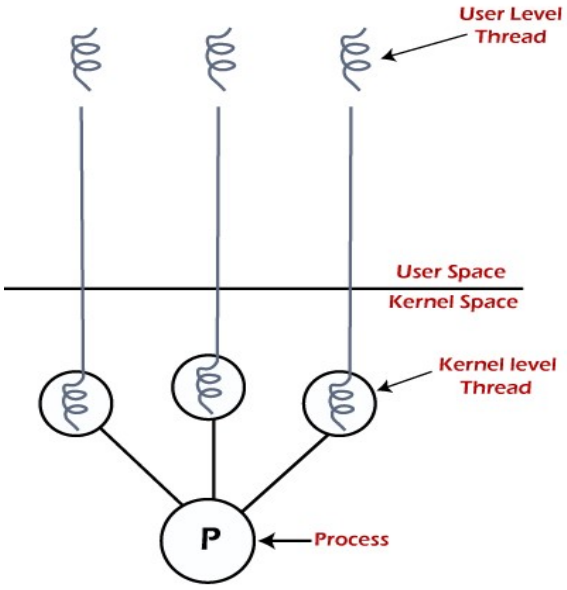
- User-level threads lack coordination between the thread and the kernel.
- If a thread causes a page fault or gets blocked, the entire process is blocked.

52

Kernel level Thread

- Threads in the kernel space designed by the OS developer to perform unique functions of OS, similar to a interrupt handler.
- The kernel thread is implemented by the operating system and hence recognized by the operating system.
- There is a thread control block and process control block in the system for each thread and process in the kernel-level thread.
- The kernel-level thread offers a system call to create and manage the threads from user-space.
- The implementation of kernel threads is more difficult than the user thread.
- Context switch time is longer in the kernel thread.
- If one kernel thread perform blocking operation then another thread can continue execution
- Example: Window Solaris.

53



54

Multithreading Models

- The user threads must be mapped to kernel threads, by one of the following strategies:
  - Many to One Model
  - One to One Model
  - Many to Many Model

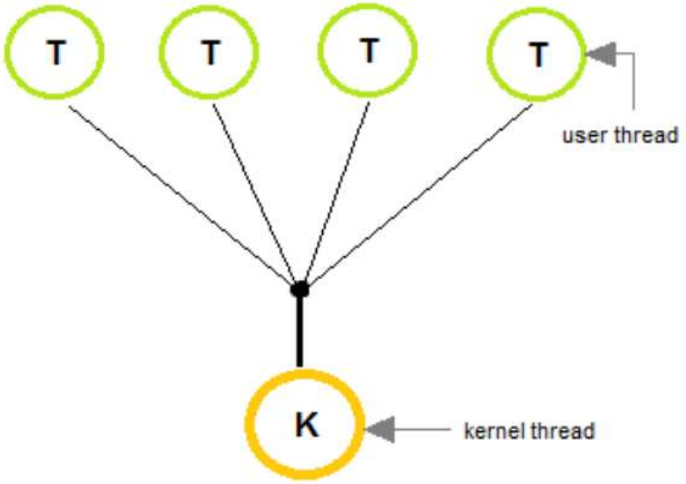
55

Many to One Model

- In the **many to one** model, many user-level threads are all mapped onto a single kernel thread.
- Thread management is handled by the thread library in user space, which is efficient in nature.
- In this case, if user-level thread libraries are implemented in the operating system in some way that the system does not support them, then the Kernel threads use this many-to-one relationship model.

56

Many to One Model



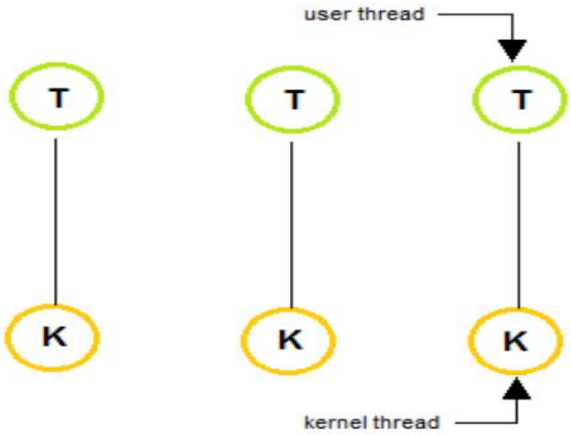
57

One to One Model

- The **one to one** model creates a separate kernel thread to handle each and every user thread.
- Most implementations of this model place a limit on how many threads can be created.
- Linux and Windows from 95 to XP implement the one-to-one model for threads.
- This model provides more concurrency than that of many to one Model.

58

One to One Model



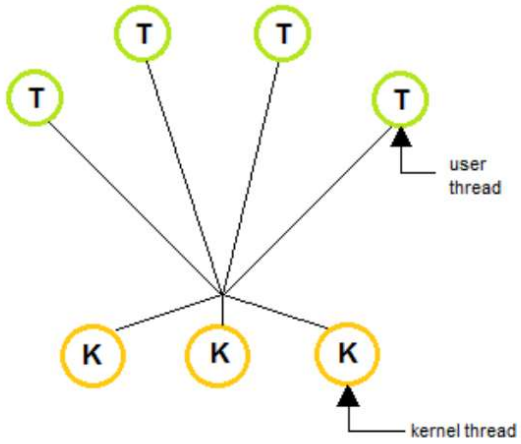
59

Many to Many Model

- The **many to many** model multiplexes any number of user threads onto an equal or smaller number of kernel threads, combining the best features of the one-to-one and many-to-one models.
- Users can create any number of threads.
- Blocking the kernel system calls does not block the entire process.
- Processes can be split across multiple processors.

60

Many to Many Model



61