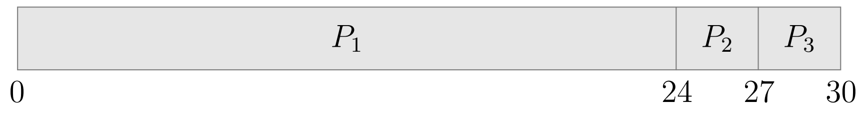
Computer Systems Tutorial

November 17th 2023

Preliminaries

Preemtive execution . Preemption is the act of temporarily interrupting an executing task, with the intention of resuming it at a later time. This interrupt is done by an external scheduler, for example, by setting a timer interrupt, or by using priorities, with no assistance or cooperation from the task.

Gantt chart A Gantt chart is a type of bar chart that illustrates a project schedule. This chart lists the tasks to be performed on the vertical axis, and time intervals on the horizontal axis. The width of the horizontal bars in the graph shows the duration of each activity. An example can be seen below.



Tasks to be performed are processes P_1, P_2, P_3, start and end time of each task are depicted on the horizontal axis. This chart shows the scheduling of these processes on a single CPU. When the number of CPUs exceeds 1, parallel processes are depicted on separate vertical lines.

Burst time A typical program consists of a sequence of interleaving I/O and CPU actions. To perform a CPU action a program needs to be allocated a CPU and the burst time refers to CPU time the current CPU action needs to execute before the next I/O action starts. In the exercises below we ignore I/O actions and hence burst time simply refers to the CPU time the process needs to complete its execution.

FCFS (first come first served) This is a simple scheduling strategy where processes are scheduled according to their position in the queue.

SJF (shortest job first) A process with the shortest burst time is scheduled first. If a program consists of several CPU actions and I/O actions the next burst time can be estimated by using the records of burst times for previous CPU actions.

Nonpreemtive priority A process with the highest priority is executed first and if another process with a higher priority arrives in the ready queue during the execution of the current process the current process keeps executing.

Preemtive priority A process with the highest priority is executed first and if another process with a higher priority arrives in the ready queue during the execution of the current process is stopped and the newly arrived high-priority process is executed next. The stopped process goes to the end of the queue.

Round Robin (RR) Processes are executed by utilising a time quantum. Once the quantum elapses, the process is stopped, and brought to the end of the queue and the next process in the queue is scheduled for the same time quantum.

Turnaround time (t/o time) The total amount of time to execute one process to its completion from the moment it first gets to the ready queue.

Waiting time The total amount of time a process has been waiting in the Ready queue. This is essentially t/o time − burst time.

CPU utilisation The percentage of time the CPU had work.

Throughput The number of processes that complete their execution per time unit.

Exercises

Exercise 1 A CPU scheduling algorithm determines an order for the execution of its scheduled processes. Given 5 processes to be scheduled on one processor, how many different schedules are possible? Assume that scheduling is nonpreemptive and no IO is made. How many different schedules are possible for n processes?

Solution We essentially need to find all the possible permutations of 5 processes (because once the process is scheduled it does not release the CPU until it finishes as we have no IO and scheduling is nonpreemptive). For the first process to schedule we have 5 options for the next one we have 4 (because we scheduled one already) and so on until we end up with only one process. This gives us $5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 120$ combinations to schedule 5 processes. For n processes the number of combinations is 5!. From a complexity point of view, we treat problems with exponential complexity to be non-tractable. But for scheduling the complexity is even worse as $\mathcal{O}(2^n) \subset \mathcal{O}(n!)$ meaning that problems with factorial complexity are harder than exponential ones. This means we need to come up with different heuristics and that a particular strategy works well only in some specific cases because it is not plausible to explore all the possible schedulings and pick the best one.

Exercise 2 Consider the following set of processes, with the length of the CPU burst time given in milliseconds:

The processes are assumed to arrive in the order listed all at time 0.

a Draw four Gantt charts that illustrate the execution of these processes using the following scheduling algorithms: FCFS, SJF, nonpreemptive priority(a larger priority number implies a higher priority), and RR(quantum=2ms).

b What is the turnaround time of each process for each of the scheduling algorithms in part a?

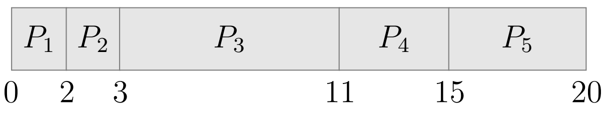
Process	Burst time	Priority
P_1	2	2
P_2	1	1
P_3	8	4
P_4	4	2
P_5	5	3

c What is the waiting time of each process for each of these scheduling algorithms?

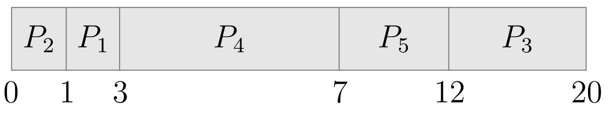
d Which of the algorithms results in the minimum average waiting time (over all processes)?

Solution

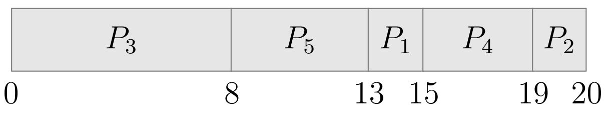
a *FCFS*



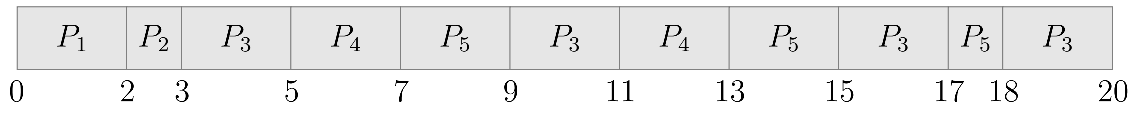
SJF



Nonpreemptive priority. Note that tie between P_1 and P_4 (they have the same priority) is broken according to FCFS.



RR



b Turnaround time

	FCFS	SJF	Priority	RR
P_1	2	3	15	2
P_2	3	1	20	3
P_3	11	20	8	20
P_4	15	7	19	13
P_5	20	12	13	18

Table 1: Turnaround time

c Waiting time

	FCFS	SJF	Priority	RR
P_1	0	1	13	0
P_2	2	0	19	2
P_3	3	12	0	12
P_4	11	3	15	9
P_5	15	7	8	13
Average	6.2	4.6	11	7.2

Table 2: Waiting time

d SJF has the smallest waiting time.

Exercise 3 The following processes are being scheduled using a preemptive, round-robin scheduling algorithm. Each process is assigned a numerical priority, with a higher number indicating a higher relative priority. In addition to the processes listed above, the system also has an **idle task** (which consumes no CPU resources and is identified as P_{idle}). This task has priority 0 and is scheduled whenever the system has no other available processes to run. The length of a time quantum is 10 units. If a process is preempted by a higher-priority process, the preempted process is placed at the end of the queue (first executes the process with the highest priority and processes with the same priority are scheduled by RR).

Process	Priority	Burst	Arrival
P_1	40	20	0
P_2	30	25	25
P_3	30	25	30
P_4	35	15	60
P_5	5	10	100
P_6	10	10	105

a Show the scheduling order of the processes using a Gantt chart.

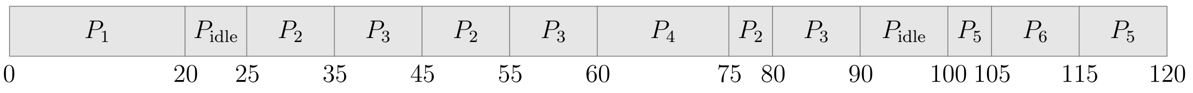
b What is the turnaround time for each process?

c What is the waiting time for each process?

d What is the CPU utilization rate (percentage of time the CPU is busy)?

Solution

a *FCFS*



b $P_1 : 20 - 0 = 20,\ P_2 : 80 - 25 = 55,\ P_3 : 90 - 30 = 60,\ P_4 : 75 - 60 = 15,\ P_5 : 120 - 100 = 20,\ P_6 :$ $115 - 105 = 10$

c $P_1 : 0,\ P_2 : 40,\ P_3 : 35,\ P_4 : 0,\ P_5 : 10,\ P_6 : 0$

d CPU utilisation is $\frac{105}{120} = 87.3\%$

Exercise 4 You and your friend John are working on a Parallel Programming project, which is an application that does not use IO functions. You have been told by your project manager that your software should have at least 6 times speedup as compared to its sequential implementation. You have also been told that your application will run on a system with 16 CPUs. Estimate the minimum percentage of the code that needs to be parallelized in your application, to meet the above performance requirement.

Solution Recall Amdahl's law

$$s = \frac{1}{(1-f) + \frac{f}{N}}$$

where f is a portion of a program that can be parallelized, N is the number of processes, and s is a theoretically achievable speed-up. Given $N = 16$ and $s \geq 6$ we have an inequality

$$\frac{1}{(1-f) + \frac{f}{16}} \geq 6$$

Solving it

$$1 \geq 6((1-f) + \frac{f}{16})$$
$$1 \geq 6 - 6f + \frac{6f}{16}$$
$$6f - \frac{6f}{16} \geq 5$$
$$5\frac{11}{16}f \geq 5$$
$$f \geq \frac{80}{91}$$

This requires f to be at least 0.87.

Exercise 5 Explain the difference between kernel threads and user threads.

Solution Kernel threads and user threads are two types of threads in the context of multitasking operating systems. They differ in terms of where they are managed and how they interact with the operating system.

Kernel Threads:

- Managed by the Kernel: Kernel threads are created and managed by the operating system's kernel. The kernel has full control over these threads.
- Operating System Support: Kernel threads are supported and scheduled by the operating system. The kernel is responsible for thread creation, scheduling, and termination.
- Concurrency: Since the kernel has direct control, kernel threads can run in parallel on multiple processors or cores, providing true parallelism.

User Threads:

- Managed by User-level Libraries: User threads are created and managed by user-level thread libraries or runtime environments. Examples include POSIX threads (pthread) and Win32 threads.
- Operating System Agnostic: The operating system may not be aware of user threads, and it may treat the entire process as a single entity. This means that user threads may not be scheduled in parallel on multiple processors.
- Concurrency: User threads rely on a single kernel thread for execution. If one user thread is blocked (e.g., waiting for I/O), it may block the entire process, including all user threads.

User threads typically use non-blocking calls to perform I/O in order to not block the process if the resource is unavailable. Please, see the following piece of Python code that uses user threads, and try to predict the output before running the program.

`https://replit.com/@andreybolkonskiy/ShySpicyKey#main.py`