CPU Scheduling

اللَّهُمَّ صَلِّ عَلَى مُحَمَّدٍ وَعَلَى آلِ مُحَمَّدٍ، كَمَا صَلَّيْتَ عَلَى إِبْرَاهِيمَ، وَبَارِكْ عَلَى مُحَمَّدٍ وَعَلَى آلِ مُحَمَّد،

كَمَا بَارَكْتَ عَلَى آلِ إِبْرَاهِيمَ، فِي الْعَالَمِينَ، إِنَّكَ حَمِيدٌ مَجِيدٌ.

By : Mohamed Gamal Maklad

- In a ***single-processor system***, **only one process can run at a time**. Other processes **must wait** until the CPU is free and can be rescheduled.
- A ***scheduling system*** allows another **process to use the I/O burst CPU when the current process has to wait for I/O**, making full use of otherwise lost CPU cycles.
- ***CPU - I/O Burst Cycle*** Process execution consists of ➔ **CPU burst followed by I/O burst**

  هنا بتكون ال processs بتنفذ حاجه➔ CPU Brust & هنا بتكون process مستنيه input او output ➔ I/O Brust

- ***CPU scheduling*** is the basis of multiprogrammed that **making the computer more productive**.
- **CPU Scheduler**:
  - ***Short-term scheduler*** **selects from among the processes in ready queue**, and **allocates the CPU to one of them**

    Queue may be ordered in various ways **هنا بنختار العمليه اللي تتنفذ من لعمليات الجاهزه لتتنفيذ**
  - **CPU scheduling decisions may take place when a process:** **ايه الاحتمالات اللي علي أساسها اختار عمليه تتنفذ**
    - When a process switches from *running to waiting* state(**nonpreemptive scheduling**)
    - When a process switches from *running to ready* state(**preemptive scheduling**)
    - When a process switches from *waiting to ready* state(**preemptive scheduling**)
    - **Terminates**(**nonpreemptive scheduling**)

هنا بيختار علي أساس ان في مثلا عمليه كانت شغاله وحصلها wait علشان مثلا محتاجه I/O او ان عمليه كانت شغاله وبعدين تحولت الي حالة ready انت شاطر و فاكر اتحولت ل ready صح ... مش فاكر طب هقولك علشان حصلها interrupt طب وحده اتحولت من wait ل ready علشان كانت محتاجه I/O والحمدلله وصل بالسلامه او ان العمليه اللي كانت شغاله دي قفلت فهيدخل بقي عمليه تانيه بدلها بقي تشتغل

- **Dispatcher :** **السي بي يو كل اللي بيعمله انه بيختار عمليه تتنفذ مين بقي اللي بيروح ينفذها هو ده بقي**
  - Dispatcher module **gives control of the CPU to the process selected** by the short-term scheduler As:
    - **switching context**
    - switching **to user mode**
    - **jumping to the proper(مناسب) location** in the user program **to restart that program**
  - ***Dispatch latency*** **time it takes for the dispatcher to stop one process and start another running**

ده بقي الوقت اللي بيضيع وانا بنقل من عمليه ل عمليه تانيه

- **Scheduling Criteria:** **اختياري للالجوريزم اللي هيختار العمليات ده معتمد علي ايه**
  - ***CPU utilization*** **keep the CPU as busy as possible** **هل جه بيحقق ان السي بي يو شغاله دايما**
  - ***Throughput*** **number of processes that complete** their **execution** per time unit **انتاجيته من حيث اكمال عدد من العمليات**
  - ***Waiting time*** **amount of time** a **process has been waiting** in the ready queue
  - ***Response time*** **amount of time it takes from when a request** was submitted until the first response is produced, not output (for i**nteractive system**)
  - ***When we use scheduling Criteria Algorithm We Want to make :***
    - **Max CPU utilization** • **Max throughput**
    - • **Min response time** • **Min turnaround time** • **Min waiting time**

❖ **How can we calculate these criteria** :

- ***Burst time*:** every process in a computer system requires some **amount of time for its execution**. This time is both the CPU time and the I/O time **ده الوقت اللي العمليه مفروض بتحتاجه علشان تتنفذ**
- ***Arrival time*** is the **time when a process enters into the ready** state and is ready for its execution
- ***Exit time (Complete time)*** is the **time when a process completes its execution** and exit from the system
- ***Response time***: the **time at which the process started for the first time on CPU - Arrival time**
  الوقت اللي العمليه فضلت مستنيه لحد ما CPU بدأ ينفذ عمليات عليها
- ***Turnaround time(ده الوقت من لحظة وصول العمليه للحظة خروجها)***: **Exit time - Arrival time**
- ***Waiting time:*** **Turnaround time - Burst time**

❖ **First- Come, First-Served (FCFS) Scheduling**: **اللي يجي الأول يخش يتنفز عليه العمليات الأول**

Consider the following set of processes that arrive at the same time and whose CPU burst length is specified in milliseconds:

Process	Burst Time
P_1	24
P_2	3
P_3	3

- **Suppose that the processes arrive in the order: PI , P2 , P The Gantt Chart for the schedule is**:نفترض ان ده ترتيب العمليات

  - Waiting time for PI = O; P2 = 24; p3=27
  - Average waiting time: (O + 24 + 27)/3 = 17➔ كده متوسط وقت الانتظار كبير

- **Suppose that the processes arrive in the order p2,p3,p1** , The Gantt chart for the schedule is:

  - Waiting time for PI = 6,p2= 0,P3= 3
  - Average waiting time: (6 +0+ 3)/3 = 3➔ كده متوسط وقت الانتظار افضل من الحل السابق

❖ **Problems with FCFS Scheduling**:

- ***It is non-preemptive algorithm***, which means the **process has been allocated to the CPU, it will never release the CPU until it finishes executing** and the process priority doesn't matter.
- **Not optimal(ليس افضل)average waiting time.**
- ***Convoy effect*** is a **situation where many processes, who need to use a resource for short time are blocked** by one **process holding that resource for a long time**. which leads to poor resource (CPU, I/O etc) utilization

هنا قولنا اللي هيجي الأول هيتنفذ الأول افرض عمليه دخلت تتنفذ احتاجت ساعه و في عمليه هتحتاج دقيقه علشان تتنفذ هتقف مستنياها ساعه علشان تخش !!! هتقولي مهي اللي جايه متأخر هقولك مش وقت احترام مواعيد لان ده طبعا يعتبر عيب لانه هيزود average time

❖ **Shortest-Job-First (SJF) Scheduling**: **هنا بقي اللي ياخد وقت تنفيذ اقل يتنفذ الأول**

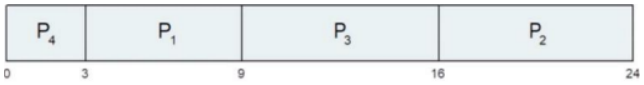
- Associate with each process the length of **its next CPU burst :** Use these lengths to **schedule the process with the shortest time**
- FCFS scheduling is used to **break a tie if the next CPU bursts** of two processes are the same.
- ***SJF is optimal*** gives **minimum average waiting** time for a given set of processes

- The difficulty **is knowing the length of the next CPU request (فممكن نخلي اليوزر هو اللي يبعته)**

بنحل هنا ازاي بيجي عندنا مدينا burst timeلكل عمليه هنختار كل عمليه تتنفذ ازاي ع أساس اللي burst time بتاعها قليل فمثلا:

Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

هنا هنفذ مثلا p4,p1,p3,p2



Avrage Waite =(3+16+9+0)/4=7

❖ Short Remining Time First:

- ***Preemptive version*** called shortest-remaining-time-first

Process	*Arrival* Time	Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

هنا بقي في المثال ملاحظين ان في arrival time

اول عمليه وصلت في 0 فكده مفيش غيرها نيجي ننفذ اول ما نوصل ل time=1 هنلاقي p2 هيكون burstبتاعها اقل فنوقف p1 بس هيكون فاضل في burst=7مش 8 ننفذ p2لما نوصل time=2 هنلاقي في عمليه بس burstبتعها اكبر من p1 فكد هنفذ p1 الأول كلها وهكذ

ف الخلاصه اننا كل ما Arrival time يزيد احنا بنعمل check هل العمليه اللي وصلت دي burstبتاعها اقل لو اه يبقي نوقف العمليه اللي احنا فيه ونخش ع العمليه اللي جت طب لو لأ و burstبتاعها اكبر هنكمل في العمليه اللي تحنا شغاليين معها وهكذا لحد ما نخلص خالص

ملاحظين هنا اهو وقفنا p1 ودخلنا علي p2 وخلصنا p2و دخلنا علي p4 علشان burstبتاعها اقل من p3و بعدين ال burstاللي في p1 =7 فكده ننفذه قبل p4

Average Wait time =((17-8)+(4-4)+(24-9)+(7-5))/4=6.5

❖ Determining Length of Next CPU Burst:

- Can **only estimate the length** should be similar to the previous one
- Then **pick process with shortest predicted next CPU burst**
- Can be done by using the length of previous CPU bursts, using ***exponential averaging***
-

1. t_n = actual length of n^{th} CPU burst
2. τ_{n+1} = predicted value for the next CPU burst
3. $\alpha, 0 \le \alpha \le 1$
4. Define: $\tau_{n+1} = \alpha\, t_n + (1-\alpha)\tau_n$.

$$\tau_{n+1} = \alpha\, t_n + (1-\alpha)\alpha\, t_{n-1} + \ldots (1-\alpha)^j \alpha t_{n-j} + \ldots + (1-\alpha)^{n+1}\, \tau_0$$

اللَّهُمَّ صَلِّ عَلَى مُحَمَّدٍ وَعَلَى آلِ مُحَمَّدٍ، كَمَا صَلَّيْتَ عَلَى إِبْرَاهِيمَ، وَبَارِكْ عَلَى مُحَمَّدٍ وَعَلَى آلِ مُحَمَّدٍ كَمَا بَارَكْتَ عَلَى آلِ إِبْرَاهِيمَ، فِي الْعَالَمِينَ، إِنَّكَ حَمِيدٌ مَجِيدٌ.

Using the **exponential averaging** and **SJF** to predict the burst time of 4th process when τ_1= 10, α = 0.5 and the previous runs are 4, 7, 8, 16.

Solution:

$$\tau_{n+1} = \alpha\, t_n + (1-\alpha)\tau_n$$

τ_2 = 0.5*4 + 0.5*10 = 7, here t_1 = 4 and τ_1 = 10

τ_3 = 0.5*7 + 0.5*7 = 7, here t_2 = 7 and τ_2 = 7

τ_4 = 0.5*8 + 0.5*7 = 7.5, here t_3 = 8 and τ_3 = 7

So the future prediction for 4th process will be τ_4 = 7.5

❖ **Priority Scheduling** : هنا بينفذ علي أساس الاولويه لكل عمليه

- **priority** number (integer) is **associated with each process**
- The CPU is **allocated to the process with the highest priority (**smallest integer ➔highest priority) that Can be :
  - Preemptive •Nonpreemptive
- SJF is priority **scheduling where priority is the inverse of predicted next CPU burst time**
- ***Problem Starvation*** —> **low** priority processes **may never execute**
- ***Solution Aging*** —> **as time progresses increase the priority of the process**

علشان ده طبعا بيتعامل مع الاولويه لو في process الاولويه بتاعتها صغيره مبتخشش و بتفضل وقفه كتير مستنيه ممكن وقت كبير هنحل المشكله دي ازاي CPU وهو معدي هيقولها ايه ي process انتي من زمان قاعده متنفذتيش ليه هنتقوله الاولويه بتاعتي كبيره ي باشا فمش راضيين ينفذوني ولا انا علشان غلباته هيقوم CPU مقلل الاولويه بتاعتها فتكون مثلا ب 2 بدل 3 فيكون لها الاولويه اكتر من الأول

Example:

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

هنا بنشوف مين اللي له priority اعلي وبعدين ننفذه بمعني اللي رقمه في priority اقل يبقي له الاولويه في التنفيذ

Priority scheduling Gantt Chart

Average waiting time = [(1-1)+(6-5)+(16-10)+(18-2)+(19-1))]/5 = 41/5 = 8.2msec

---

❖ **Example of Priority Preemptive**:

Turnaround time = **Exit time — Arrival time**

Waiting time = ***Turnaround time*** — **Burst time**

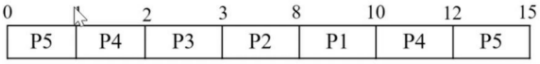
Response time = **start time - Arrival time**

Process	Arrival time	Burst time	Priority
P1	4	2	1
P2	3	5	1
P3	2	1	3
P4	1	3	5
P5	0	4	6

هنا بنشوف اول عمليه وصلت بيكون arrival time =0

اول ما يتغير arrival time =1 بنشوف هل العمليه اللي جت دي ليها الاولوسه انها تتنفذ لو اه يبقي وقف العمليه اللي شغاله دلوقتي وخش علي اللي ليها الاولويه طب لو لأ كمل تنفيذ العمليه اللي انت فيها دلوقتي

خلصنا وكله تمام نبدأ نحسب turnaround time , waiting time , response time



Process Id	Exit time	Turnaround time	Waiting time	Response time
P1	10	10 – 4 = 6	6 – 2 = 4	8 – 4 = 4
P2	8	8 – 3 = 5	5 – 5 = 0	3 – 3 = 0
P3	3	3 – 2 = 1	1 – 1 = 0	2 – 2 = 0
P4	12	12 – 1 = 11	11 – 3 = 8	1 – 1 = 0
P5	15	15 – 0 = 15	15 – 4 = 11	0 – 0 = 0

اللَّهُمَّ صَلِّ عَلَى مُحَمَّدٍ وَعَلَى آلِ مُحَمَّدٍ، كَمَا صَلَّيْتَ عَلَى إِبْرَاهِيمَ، وَبَارِكْ عَلَى مُحَمَّدٍ وَعَلَى آلِ مُحَمَّدٍ كَمَا بَارَكْتَ عَلَى آلِ إِبْرَاهِيمَ، فِي الْعَالَمِينَ، إِنَّكَ حَمِيدٌ مَجِيدٌ

Average Turnaround time = (6 + 5 + 1 + 11 + 15) / 5 = 38 / 5 = 7.6 unit

Average waiting time = (4 + 0 + 0 + 8 + 11) / 5 = 23 / 5 = 4.6 unit

Average response time = (4 + 0 + 0 + 0 + 0) / 5 = 4 / 5 = 0.8 unit

❖ Round Robin (RR) :

- Each process gets a small unit of CPU time (**time quantum q**), usually 10-100 milliseconds. **After this time has elapsed, the process is preempted and added to the end of the ready queue**

بمعني ان بيكون متحدد وقت معين للتنفيذ طب لو الوقت خلص قبل ما العمليه تخلص بنحطها في الاخرqueue ونفذ العمليه اللي عليها الدور

- **RR scheduling is similar to FCFS scheduling**, except that CPU bursts are assigned with **limits called** time **quantum** (time slice).
- The RR scheduling algorithm is **designed especially for timesharing systems**. It gives the best performance in terms of average response time.
- If there are n processes in the ready queue and the time quantum is q, **then each process gets l/n of the CPU time**. **No process waits more than (n-l)q time** units until its next time quantum
- *Example on RR with time quantum =4*

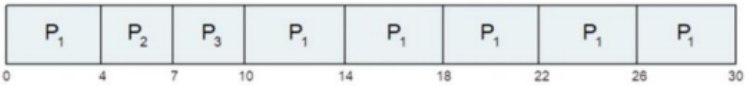
Process	Burst Time
P_1	24
P_2	3
P_3	3

هو هنا حدد time quantum=4

كده كل عمليه هتنفذ 4 بس من bursted time الخاص بيها

فهنا نفذ 4 من p1 وبعدين p2,p3, وبعدين 4 من p1 وهكذا بيحيث ان كل عمليه ليها quantum time 4

The Gantt chart is:

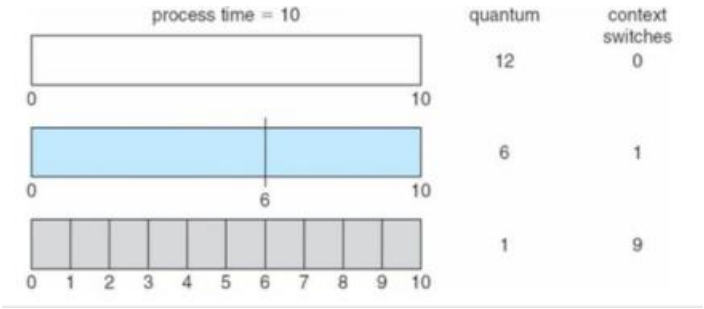


Average waiting time = [(30-24)+(7-3)+(10-3)]/3 = 17/3 = 5.6

- The **average waiting time under the RR policy is often long**.
- Typically, **higher average turnaround than SJF, but better response**

❖ time quantum Size affect in Round Robin (RR):

- The **performance of the RR depends heavily on the size of q**
  - ***q large*** o if the time **quantum is extremely large, the RR policy is FCFS policy**.
  - ***q small***
    - if the time quantum is extremely small (say, 1 millisecond), **the RR approach can result in a large number of context switches**
    - q must be large with respect to context switch, otherwise overhead is too



جه في بالك دلوقتي قولت ما انا اخلي time quantum ويلف علي كل process كله كل شويه كل واحده جزء صغير من الوقت هقولك انت شكلك نسيت صح ..... نسيت اسمع كلامي هو مش لما يبقي في time quantum صغير معني كده هنبدل ع العمليات كتير هتقولي اه مهو حلو خقولك حلو ازاي هوي مش كل عملية تحويل من عمليه للتانيه ده بيعوز Context switch وده بياخد وقت فبدل ما هوفر وقت لا دا انا هضيع وقت اكتر في context switch

❖ Multilevel Queue : ده بقي بيعمل ميكس بين الأنواع اللي فاتت

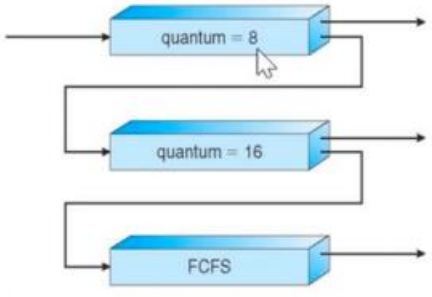
- ***Ready queue*** is partitioned into separate queues, eg:
  - **foreground (interactive)**
  - **background (batch)**
- **Process permanently** in a given queue
- **Each queue has its own scheduling algorithm**: for example :
  - foreground — RR
  - background — FCFS
- Scheduling must be done between the queues:
  - ***Fixed priority*** scheduling; (i.e., **serve all from foreground then from background**). Possibility of **starvation**. **بخلص الدور اللي فوق الأول و بعد كده الدور التاني**
  - ***Time slice*** **each queue gets a certain amount of CPU time** which it can schedule amongst its processes; ***هنا بقي بيقسم الوقت شويه هنا و شويه هنا***

❖ Multilevel Feed Back Queue : هنا ممكن البروسيس تتنقل من كيو الي كيو تاني او من طبقه للتانيه

- A **process can move between the various queues**; aging can be implemented this way
- **Multilevel-feedback**-queue scheduler defined by the following parameters:
  - **number of queues**
  - **scheduling algorithms** for each queue
  - method used to determine ***when to upgrade a process***
  - method used to determine ***when to demote a process***
  - method used to determine ***which queue a process will enter when that process needs service***

❖ Example of Multilevel Feedback Queue :

هنا العمليه بتاعتنا اول ما دخلت دخلناها علي اول ليفل اللي هو فيه RR و اللي time quantum بتاعها ب 8 دخلت البروسيس الوقت خلص وهي مخلصتش قولتلها لا انت تنزلي تحت شكلك مطوله بدل نزلتها تحت بيقي كده انا عملتلها demote بعد كده نزلت layer اللي تحت برضه فيها RR ال time quantum 16 الوقت عدي و البروسيس مخلصتش ثولتها انزلي تحت بقي انت شكلك مطوله ف كده انا نقلت ال process ما بين كذا layer فكده حققت feed back queue



By : Mohamed Gamal Maklad