جامعة المنارة
كلية الهندسة
قسم الهندسة المعلوماتية

مقرر نظم التشغيل 1

خوارزميات جدولة المعالج

أ.د. جمال خليفة

م.جمال محمود – م.جهاد عيسى

جلسة الأسبوع الخامس والسادس

(الفصل الثاني 2023 | 2024)

الغاية من الجلسة

- ✓ التعريف بخوارزميات جدولة وحدة المعالجة المركزية CPU Scheduling.
- ✓ تنفيذ خوارزميات الجدولة باستخدام لغة C++.
- ✓ حل بعض النماذج الرياضية لخوارزميات الجدولة.

مقدمة

تستخدم أنظمة تشغيل Windows، بما في ذلك Windows 11، عادةً مجموعة من خوارزميات الجدولة لإدارة موارد وحدة المعالجة المركزية بكفاءة. من أهم خوارزميات الجدولة الأساسية المستخدمة في أنظمة التشغيل الحديثة هي:

- **First-Come-First-Served من يأتي أولاً يُخدم أولاً (FCFS):** تقوم خوارزمية FCFS بجدولة العمليات بترتيب وصولها. إنه نهج بسيط ومباشر، ولكنه يمكن أن يؤدي إلى فترات انتظار طويلة للعمليات ذات أوقات تنفيذ أعلى، مما يؤدي إلى عدم الكفاءة المحتملة.
- **Shortest Job Next المهمة الأقصر هي التالية (SJN):** تعطي خوارزمية SJN الأولوية للعمليات ذات أقصر وقت تنفيذ متوقع. يهدف هذا الأسلوب إلى تقليل متوسط وقت الانتظار للعمليات عن طريق تنفيذ أقصر مهمة أولاً.
- **Priority Scheduling جدولة الأولوية:** تقوم جدولة الأولوية بتعيين قيمة أولوية لكل عملية، ويتم تخصيص وحدة المعالجة المركزية للعملية ذات الأولوية القصوى. تضمن هذه الخوارزمية أن العمليات الحرجة أو ذات الأولوية العالية تتلقى وقت وحدة المعالجة المركزية عند الحاجة، ولكنها قد تؤدي إلى إهمال العمليات ذات الأولوية المنخفضة.
- **Round Robin الشريط الدوار:** تقوم خوارزمية Round Robin بتعيين شريحة زمنية أو كمية ثابتة لكل عملية وتدور خلال العمليات بطريقة دائرية. تضمن هذه الخوارزمية التوزيع العادل لوقت وحدة المعالجة المركزية بين العمليات وتساعد على منع أي عملية منفردة من احتكار وحدة المعالجة المركزية.

والعديد من الخوارزميات الأخرى .....

خوارزمية من يأتي أولاً يُخدم أولاً First-Come-First-Served (FCFS):

وهي أبسط خوارزميات جدولة المعالج، حيث تعتمد جدولة التنفيذ بحسب زمن وصول العملية. تنص خوارزمية الجدولة FCFS على أن العملية التي تطلب وحدة المعالجة المركزية أولاً يتم تخصيصها لوحدة المعالجة المركزية أولاً. تم تنفيذ هذه الخوارزمية باستخدام FIFO Queue. عندما تدخل عملية ما إلى قائمة الانتظار الجاهزة يتم توضعها في ذيل قائمة الانتظار، وعندما تصبح وحدة المعالجة المركزية حرة، يتم دفع العملية إلى رأس قائمة الانتظار وخدمتها ثم تتم إزالة العملية الجارية من قائمة الانتظار.

خصائص FCFS

- يتم تنفيذ المهام دائمًا وفقًا لمفهوم أسبقية الحضور.
- FCFS سهلة التنفيذ والاستخدام.
- هذه الخوارزمية ليست فعالة جدًا في الأداء، ووقت الانتظار طويل جدًا.

خوارزمية الجدولة FCFS

- وقت انتظار العملية الأولى هو 0 حيث يتم تنفيذها أولاً.
- يمكن حساب وقت الانتظار للعملية القادمة من العلاقة:

$$wt[i] = (at[i-1] + bt[i-1] + wt[i-1]) - at[i]$$

- wt[i] زمن الانتظار للعملية الحالية
- at[i-1] زمن الوصول للعملية السابقة
- bt[i-1] زمن التنفيذ للعملية السابقة
- wt[i-1] زمن الانتظار للعملية السابقة
- at[i] زمن الوصول للعملية الحالية

- يمكن حساب زمن الانتظار الوسطي من العلاقة:

Average Waiting Time = (sum of all waiting time)/(Number of processes)

أي :

زمن الانتظار الوسطي = مجموع أزمنة انتظار جميع العمليات / عدد العمليات

نبين فيما يلي مثالاً توضح آلية عمل هذه الخوارزمية

ليكن لدينا الجدول التالي الذين يبن أزمنة الوصول والتنفيذ لخمس عمليات P1,P2,P3,P4,P5 :

Processes	Arrival Time	Burst Time
P1	**0**	**4**
P2	**1**	**3**
P3	**2**	**1**
P4	**3**	**2**
P5	**4**	**5**

ستنفذ هذه الخوارزمية على خطوات كما يلي:

الخطوة 0 ، في اللحظة 0 :

يبدأ تنفيذ العملية P1 باعتبار أنها وصلت في اللحظة 0

Time Instance	Process	Arrival Time	Waiting Table	Execution Time	Initial Burst Time	Remaining Burst Time
0-1ms	**P1**	**0ms**		**1ms**	**4ms**	**3ms**

الخطوة 1 ، في اللحظة 1 :

تصل العملية P2 لكن العملية P1 لازالت بحالة تنفيذ وبالتالي ستبقى العملية P2 في جدول الانتظار إلى أن يحين موعد تنفيذها.

Time Instance	Process	Arrival Time	Waiting Table	Execution Time	Initial Burst Time	Remaining Burst Time
1-2ms	**P1**	**0ms**		**1ms**	**3ms**	**2ms**
	P2	**1ms**	**P2**	**0ms**	**3ms**	**3ms**

الخطوة 3، في اللحظة 2 :

تصل العملية P3 لكن العملية P1 لازالت بحالة تنفيذ وبالتالي تبقى العملية P3 في جدول الانتظار.

Time Instance	Process	Arrival Time	Waiting Table	Execution Time	Initial Burst Time	Remaining Burst Time
2-3ms	P1	0ms		1ms	2ms	1ms
	P2	1ms	P2	0ms	3ms	3ms
	P3	2ms	P2, P3	0ms	1ms	1ms

الخطوة 4، في اللحظة 3 :

تصل العملية P4 لكن العملية P1 لازالت بحالة تنفيذ وبالتالي ستبقى العملية P4 في جدول الانتظار إلى أن يحين موعد تنفيذها.

Time Instance	Process	Arrival Time	Waiting Table	Execution Time	Initial Burst Time	Remaining Burst Time
3-4ms	P1	0ms		1ms	1ms	0ms
	P2	1ms	P2	0ms	3ms	3ms
	P3	2ms	P2, P3	0ms	1ms	1ms
	P4	3ms	P2, P3, P4	0ms	2ms	2ms

الخطوة 5، في اللحظة 4 :

تنهي العملية P1 تنفيذها، وتصل العملية P5 إلى جدول الانتظار حيث يبدأ تنفيذ العملية P2.

Time Instance	Process	Arrival Time	Waiting Table	Execution Time	Initial Burst Time	Remaining Burst Time
4-5ms	P2	1ms		1ms	3ms	2ms
	P3	2ms	P3	0ms	1ms	1ms
	P4	3ms	P3, P4	0ms	2ms	2ms
	P5	4ms	P3, P4, P5	0ms	5ms	5ms

الخطوة 6، في اللحظة 5 :

تتابع العملية P2 تنفيذها وتبقى العمليات P3,P4,P5 في جدول الانتظار.

Time Instance	Process	Arrival Time	Waiting Table	Execution Time	Initial Burst Time	Remaining Burst Time
5-7ms	P2	1ms		2ms	2ms	0ms
	P3	2ms	P3	0ms	1ms	1ms
	P4	3ms	P3, P4	0ms	2ms	2ms
	P5	4ms	P3, P4, P5	0ms	5ms	5ms

الخطوة 7، في اللحظة 7 :

تنهي العملية P2 تنفيذها ويبدأ تنفيذ العملية P3 وباعتبار زمن تنفيذها هو 1 وبالتالي تنهي تنفيذها في اللحظة 8.

Time Instance	Process	Arrival Time	Waiting Table	Execution Time	Initial Burst Time	Remaining Burst Time
7-8ms	P3	2ms		1ms	1ms	0ms
	P4	3ms	P4	0ms	2ms	2ms
	P5	4ms	P4, P5	0ms	5ms	5ms

الخطوة 8، في اللحظة 8:

تنهي العملية P3 تنفيذها ويبدأ تنفيذ العملية P4 وباعتبار زمن تنفيذها هو 2 وبالتالي تنهي تنفيذها في اللخظة 10.

Time Instance	Process	Arrival Time	Waiting Table	Execution Time	Initial Burst Time	Remaining Burst Time
8-10ms	P4	3ms		2ms	2ms	0ms
	P5	4ms	P5	0ms	5ms	5ms

الخطوة 9، في اللحظة 10:

تنهي العملية P4 تنفيذها ويبدأ تنفيذ العملية P5 وباعتبار زمن تنفيذها هو 5 وبالتالي تنهي تنفيذها في اللحظة 15.

Time Instance	Process	Arrival Time	Waiting Table	Execution Time	Initial Burst Time	Remaining Burst Time
10-15ms	P5	4ms		5ms	5ms	0ms

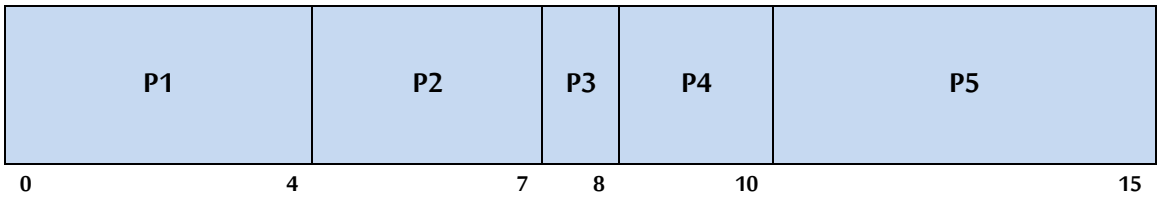
الخطوة 10، في اللحظة 15:

تنهي العملية P5 وبالتالي تكون الخلاصة الكلية للتنفيذ على النحو:

Time Instance	Process	Arrival Time	Waiting Table	Execution Time	Initial Burst Time	Remaining Burst Time
0-1ms	P1	0ms		1ms	4ms	3ms
1-2ms	P1	0ms		1ms	3ms	2ms
	P2	1ms	P2	0ms	3ms	3ms
2-3ms	P1	0ms		1ms	2ms	1ms
	P2	1ms	P2	0ms	3ms	3ms
	P3	2ms	P2, P3	0ms	1ms	1ms
3-4ms	P1	0ms		1ms	1ms	0ms
	P2	1ms	P2	0ms	3ms	3ms
	P3	2ms	P2, P3	0ms	1ms	1ms
	P4	3ms	P2, P3, P4	0ms	2ms	2ms
4-5ms	P2	1ms		1ms	3ms	2ms
	P3	2ms	P3	0ms	1ms	1ms
	P4	3ms	P3, P4	0ms	2ms	2ms
	P5	4ms	P3, P4, P5	0ms	5ms	5ms
5-7ms	P2	1ms		2ms	2ms	0ms

	P3	2ms	P3	0ms	1ms	1ms
	P4	3ms	P3, P4	0ms	2ms	2ms
	P5	4ms	P3, P4, P5	0ms	5ms	5ms
7-8ms	P3	2ms		1ms	1ms	0ms
	P4	3ms	P4	0ms	2ms	2ms
	P5	4ms	P4, P5	0ms	5ms	5ms
8-10ms	P4	3ms		2ms	2ms	0ms
	P5	4ms	P5	0ms	5ms	5ms
10-15ms	P5	4ms		5ms	5ms	0ms

مخطط غانت Gantt لعملية التنفيذ السابقة :



حساب أزمنة الانتظار للعمليات وزمن الانتظار الوسطي:

Waiting Time = Start time – Arrival time

P1 = 0 – 0 = 0

P2 = 4 – 1 = 3

P3 = 7 – 2 = 5

P4 = 8 – 3 = 5

P5 = 10 – 4 = 6

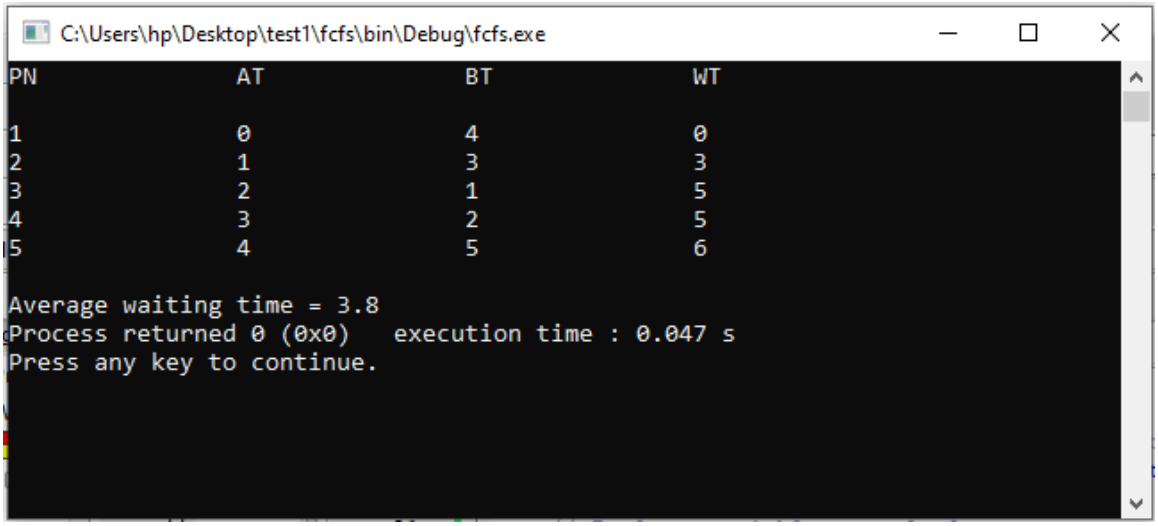
Average waiting time = (0 + 3 + 5 + 5+ 6)/ 5 = 19 / 5 = 3.8

تحقيق الخوارزمية بلغة C++

```
// C++ program to Calculate Waiting Time for given Processes
#include <iostream>
using namespace std;
// Function to Calculate waiting time and average waiting time
void CalculateWaitingTime(int at[], int bt[], int N)
{
        // Declare the array for waiting time
        int wt[N];
        // Waiting time for first process is 0
        wt[0] = 0;
        // Print waiting time process 1
        cout << "PN\t\tAT\t\t"<< "BT\t\tWT\n\n";
        cout << "1" << "\t\t" << at[0] << "\t\t" << bt[0] << "\t\t" << wt[0] << endl;
        // Calculating waiting time for each process from the given formula
        for (int i = 1; i < 5; i++) {
                wt[i] = (at[i - 1] + bt[i - 1]+ wt[i - 1]) - at[i];
                // Print the waiting time for each process
                cout << i + 1 << "\t\t" << at[i] << "\t\t" << bt[i] << "\t\t" << wt[i] << endl;
        }
        // Declare variable to calculate average
        float average;
        float sum = 0;
        // Loop to calculate sum of all waiting time
        for (int i = 0; i < 5; i++) {
                sum = sum + wt[i];
        }
```

```
        // Find average waiting time by dividing it by no. of process
        average = sum / 5;
        // Print Average Waiting Time
        cout << "\nAverage waiting time = " << average;
}
// Driver code
int main()
{
        int N = 5;          // Number of process
        int at[] = { 0, 1, 2, 3, 4 };  // Array for Arrival time
        int bt[] = { 4, 3, 1, 2, 5 };  // Array for Burst Time
        // Function call to find waiting time
        CalculateWaitingTime(at, bt, N);
        return 0;
}
```

يبين الشكل ناتج تنفيذ هذا البرنامج:



خوارزمية المهمة الأقصر هي التالية Shortest Job Next (SJN):

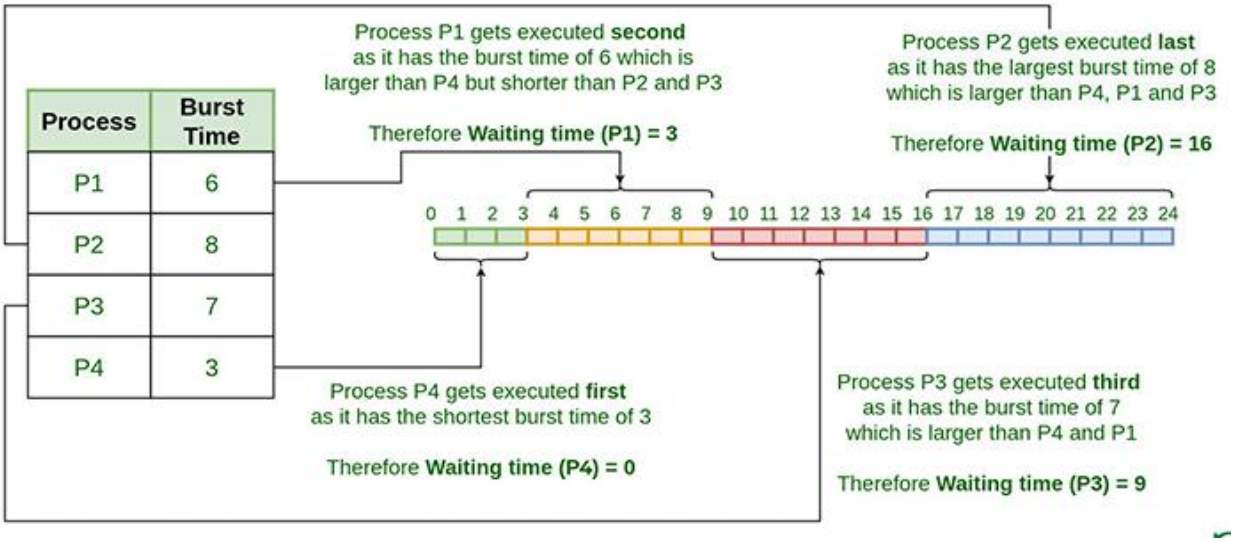
خوارزمية المهمة الأقصر هي التالية (Shortest Job Next (SJN (ويطلق عليها أحياناً المهمة الأقصر أولاً Shortest Job First (SJF)) هي خوارزمية جدولة تحدد عملية الانتظار مع أصغر وقت تنفيذ للتنفيذ بعد ذلك.

خصائص خوارزمية الجدولة SJN:

- تتميز هذه الخوارزمية بأنها تحقق الحد الأدنى لمتوسط وقت الانتظار بين جميع خوارزميات الجدولة.
- هذه الخوارزمية جشعة Greedy وبالتالي قد تسبب استعصاء أو تأخراً في تنفيذ العمليات الطويلة إذا استمر قدوم العمليات الأقصر
- تكون عموماً غير ممكنة الاستخدام في الحالات التي لا يستطيع فيها نظام التشغيل التنبؤ بزمن التنفيذ لعملية ما وبالتالي يصغب فرز العمليات حسب الأقصر.

خوارزمية الجدولة SJN:

- يتم فرز جميع العمليات وفقا لوقت الوصول.
- ثم يتم تحديد العملية التي تملك الحد الأدنى من وقت الوصول والحد الأدنى من وقت التنفيذ.
- بعد الانتهاء من العملية، يتم إنشاء مجموعة من العمليات التي تصل بعد ذلك حتى الانتهاء من العملية السابقة وتحديد العملية من بين المجموعة التي تتمتع بالحد الأدنى من وقت التنفيذ.



الحسابات الزمنية في خوارمية SJN

- **زمن الإكمال Completion Time:** الزمن الذي تنتهي فيه العملية من تنفيذها.
- **زمن الدوران Turn Around Time:** فارق الوقت بين زمن الانتهاء وزمن الوصول.

زمن الدوران = زمن الانتهاء – زمن الوصول

- **زمن الانتظار(W.T):** فرق الوقت بين زمن الدوران وزمن التنفيذ.

زمن الانتظار= زمن الدوران – زمن التنفيذ

نبين فيما يلي مثالاً توضح آلية عمل هذه الخوارزمية

ليكن لدينا الجدول التالي الذين يبن أزمنة الوصول والتنفيذ لخمس عمليات P1,P2,P3,P4,P5 :

Processes	Arrival Time	Burst Time
P1	**2ms**	**6ms**
P2	**5ms**	**2ms**
P3	**1ms**	**8ms**
P4	**0ms**	**3ms**
P5	**4ms**	**4ms**

ستنفذ هذه الخوارزمية على خطوات كما يلي:

في اللحظة 0 :

تصل العملية P4 وتبدأ بالتنفيذ

Time Instance	Process	Arrival Time	Waiting Table	Execution Time	Initial Burst Time	Remaining Burst Time
0-1ms	**P4**	**0ms**		**1ms**	**3ms**	**2ms**

في اللحظة 1 :

تصل العملية P3 ولكن بما أن العملية P4 لم تنته بعد وبالتالي ستنتظر العملية P3 لحين انتهاء P4 من التنفيذ.

Time Instance	Process	Arrival Time	Waiting Table	Execution Time	Initial Burst Time	Remaining Burst Time
1-2ms	P4	0ms		1ms	2ms	1ms
	P3	1ms	P3	0ms	8ms	8ms

في اللحظة 2 :

تصل العملية P1 وتضاف إلى جدول الانتظار وتستمر P4 في التنفيذ.

Time Instance	Process	Arrival Time	Waiting Table	Execution Time	Initial Burst Time	Remaining Burst Time
2-3ms	P4	0ms		1ms	1ms	0ms
	P3	1ms	P3	0ms	8ms	8ms
	P1	2ms	P3, P1	0ms	6ms	6ms

في اللحظة 3 :

ستنهي العملية P4 تنفيذها، وتتم مقارنة زمن تنفيذ العمليتين P3 و P1 الموجودتين في جدول الانتظار وباعتبار زمن تنفيذ P1 أقل سيتم البدء بتنفيذها.

Time Instance	Process	Arrival Time	Waiting Table	Execution Time	Initial Burst Time	Remaining Burst Time
3-4ms	P3	1ms	P3	0ms	8ms	8ms
	P1	2ms	P3	1ms	6ms	5ms

في اللحظة 4 :

تصل العملية P5 وتنضم إلى جدول الانتظار وتستمر P1 في التنفيذ.

Time Instance	Process	Arrival Time	Waiting Table	Execution Time	Initial Burst Time	Remaining Burst Time
4-5ms	P3	1ms	P3	0ms	8ms	8ms
	P1	2ms	P3	1ms	5ms	4ms
	P5	4ms	P3, P5	0ms	4ms	4ms

في اللحظة 5 :

تصل العملية P2 وتنضم إلى جدول الانتظار وتستمر P1 في التنفيذ.

Time Instance	Process	Arrival Time	Waiting Table	Execution Time	Initial Burst Time	Remaining Burst Time
5-6ms	P3	1ms	P3	0ms	8ms	8ms
	P1	2ms	P3	1ms	4ms	3ms
	P5	4ms	P3, P5	0ms	4ms	4ms
	P2	5ms	P3, P5, P2	0ms	2ms	2ms

في اللحظة 6 :

تنهي العملية P1 ويتم مقارنة أزمنة تنفيذ العمليات P3,P5,P2 وباعتبار زمن تنفيذ P2 أقل سيتم البدء بتنفيذها.

Time Instance	Process	Arrival Time	Waiting Table	Execution Time	Initial Burst Time	Remaining Burst Time
6-9ms	P3	1ms	P3	0ms	8ms	8ms
	P1	2ms	P3	3ms	3ms	0ms
	P5	4ms	P3, P5	0ms	4ms	4ms
	P2	5ms	P3, P5, P2	0ms	2ms	2ms

في اللحظة 9 :

يتم تنفيذ العملية P2 وتبقى P3 و P5 في جدول الانتظار.

Time Instance	Process	Arrival Time	Waiting Table	Execution Time	Initial Burst Time	Remaining Burst Time
9-11ms	P3	1ms	P3	0ms	8ms	8ms
	P5	4ms	P3, P5	0ms	4ms	4ms
	P2	5ms	P3, P5	2ms	2ms	0ms

في اللحظة 11 :

ينتهي تنفيذ العملية P2 ويتم مقارنة أزمنة تنفيذ العمليات P3 و P5 وباعتبار زمن تنفيذ P5 أقل سيتم البدء بتنفيذها.

Time Instance	Process	Arrival Time	Waiting Table	Execution Time	Initial Burst Time	Remaining Burst Time
11-15ms	P3	1ms	P3	0ms	8ms	8ms
	P5	4ms	P3	4ms	4ms	0ms

في اللحظة 15 :

ينتهي تنفيذ العملية P5

Time Instance	Process	Arrival Time	Waiting Table	Execution Time	Initial Burst Time	Remaining Burst Time
15-23ms	P3	1ms		8ms	8ms	0ms

في اللحظة 23 :

ينتهي تنفيذ العملية P3 وسيكون جدول التنفيذ الإجمالي على النحو:

Time Instance	Process	Arrival Time	Waiting Table	Execution Time	Initial Burst Time	Remaining Burst Time
0-1ms	P4	0ms		1ms	3ms	2ms
1-2ms	P4	0ms		1ms	2ms	1ms
	P3	1ms	P3	0ms	8ms	8ms
2-3ms	P4	0ms		1ms	1ms	0ms
	P3	1ms	P3	0ms	8ms	8ms
	P1	2ms	P3, P1	0ms	6ms	6ms
3-4ms	P3	1ms	P3	0ms	8ms	8ms
	P1	2ms	P3	1ms	6ms	5ms
4-5ms	P3	1ms	P3	0ms	8ms	8ms
	P1	2ms	P3	1ms	5ms	4ms
	P5	4ms	P3, P5	0ms	4ms	4ms
5-6ms	P3	1ms	P3	0ms	8ms	8ms
	P1	2ms	P3	1ms	4ms	3ms
	P5	4ms	P3, P5	0ms	4ms	4ms
	P2	5ms	P3, P5, P2	0ms	2ms	2ms
6-9ms	P3	1ms	P3	0ms	8ms	8ms
	P1	2ms	P3	3ms	3ms	0ms
	P5	4ms	P3, P5	0ms	4ms	4ms
	P2	5ms	P3, P5, P2	0ms	2ms	2ms
9-11ms	P3	1ms	P3	0ms	8ms	8ms
	P5	4ms	P3, P5	0ms	4ms	4ms
	P2	5ms	P3, P5	2ms	2ms	0ms

11-15ms	P3	1ms	P3	0ms	8ms	8ms
	P5	4ms	P3	4ms	4ms	0ms
15-23ms	P3	1ms		8ms	8ms	0ms

مخطط غانت Gantt لعملية التنفيذ السابقة :

P4	P1	P2	P5	P3
0 3	9	11	15	23

حساب زمن الانتظار الوسطي:

P4 = 0 – 0 = 0

P1 = 3 – 2 = 1

P2 = 9 – 5 = 4

P5 = 11 – 4 = 7

P3 = 15 – 1 = 14

Average Waiting Time = 0 + 1 + 4 + 7 + 14/5 = 26/5 = 5.2

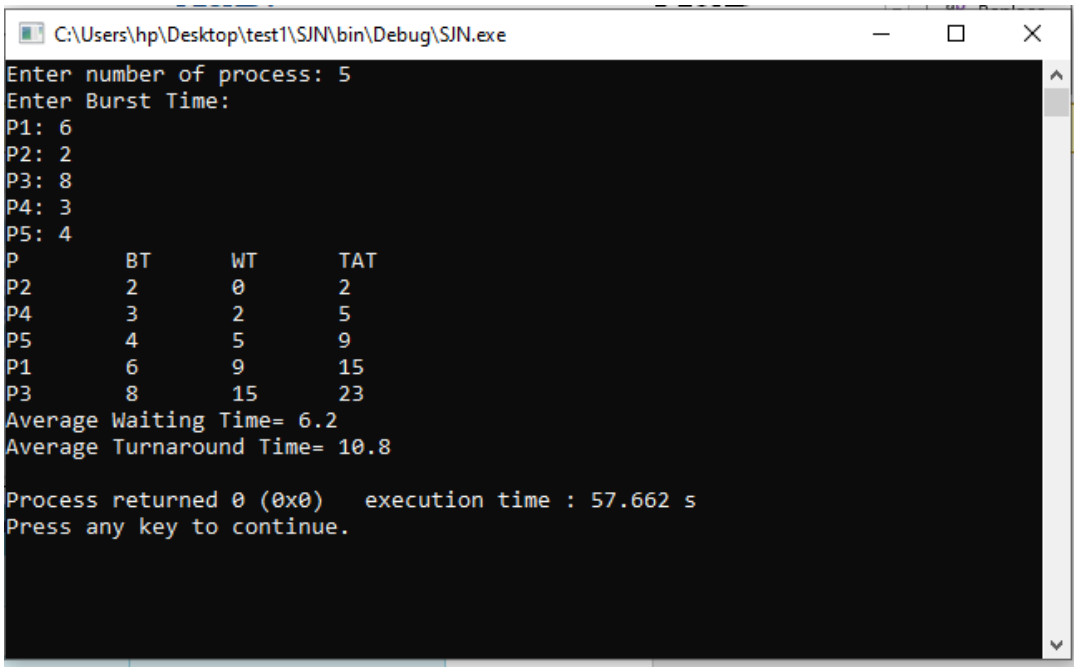
تحقيق الخوارزمية بلغة C++

```
#include <iostream>
using namespace std;
int main() {
// Matrix for storing Process Id, Burst Time, Average Waiting Time & Average Turn Around Time.
        int A[100][4];
        int i, j, n, total = 0, index, temp;
        float avg_wt, avg_tat;
        cout << "Enter number of process: ";
```

```
cin >> n;
cout << "Enter Burst Time:" << endl;
// User Input Burst Time and alloting Process Id.
for (i = 0; i < n; i++) {
        cout << "P" << i + 1 << ": ";
        cin >> A[i][1];
        A[i][0] = i + 1;
}
// Sorting process according to their Burst Time.
for (i = 0; i < n; i++) {
        index = i;
        for (j = i + 1; j < n; j++)
                if (A[j][1] < A[index][1])
                        index = j;
        temp = A[i][1];
        A[i][1] = A[index][1];
        A[index][1] = temp;
        temp = A[i][0];
        A[i][0] = A[index][0];
        A[index][0] = temp;
}
A[0][2] = 0;
// Calculation of Waiting Times
for (i = 1; i < n; i++) {
        A[i][2] = 0;
        for (j = 0; j < i; j++)
                A[i][2] += A[j][1];
        total += A[i][2];
```

```
        }
        avg_wt = (float)total / n;
        total = 0;
        cout << "P        BT        WT        TAT" << endl;
        // Calculation of Turn Around Time and printing the data.
        for (i = 0; i < n; i++) {
                A[i][3] = A[i][1] + A[i][2];
                total += A[i][3];
        cout << "P" << A[i][0] << "          " << A[i][1] << "  " << A[i][2] << "  " << A[i][3] << endl;
        }
        avg_tat = (float)total / n;
        cout << "Average Waiting Time= " << avg_wt << endl;
        cout << "Average Turnaround Time= " << avg_tat << endl;
}
```

يبين الشكل ناتج تنفيذ هذا البرنامج:



خوارزمية الأولوية Priority Scheduling:

تعد جدولة الأولوية من خوارزميات الجدولة الأكثر شيوعًا في أنظمة التشغيل. يتم تعيين وقت وصول أول لكل عملية (العملية ذات وقت الوصول الأقل أولاً) إذا كانت هناك عمليتان لهما نفس وقت الوصول، تتم مقارنتهما بالأولويات (العملية الأعلى أولاً). وأيضًا، إذا كانت هناك عمليتان لهما نفس الأولوية، تتم مقارنتهما برقم العملية (أقل من رقم العملية أولاً). يتم تكرار هذه العملية أثناء تنفيذ كافة العمليات.

ألية عمل الخوارزمية

- يتم أولاً إدخال العمليات مع وقت وصولها ووقت التنفيذ والأولوية.
- ستتم جدولة العملية الأولى، والتي لها أقل وقت وصول، إذا كانت عمليتان أو أكثر لها أقل وقت وصول، فإن من لديه أولوية أعلى سيقوم بالجدولة أولاً.
- الآن سيتم جدولة المزيد من العمليات وفقًا لوقت الوصول وأولوية العملية. (هنا نفترض أن انخفاض رقم الأولوية له أولوية أعلى). إذا كانت أولوية العملية متماثلة، فقم بالفرز وفقًا لرقم العملية.
- بمجرد وصول جميع العمليات، يمكننا جدولتها بناءً على أولويتها.

ملاحظات:

1. في تساوي الأولويات يتم التنفيذ وفق خوارزمية القادم أولاً ينفذ أولاً FCFS.
2. في حال تساوي أزمنة الوصول، يتم ترتيب العمليات حسب ترتيب الأولوية وأيضاً تنفيذها وفق هذا الترتيب.

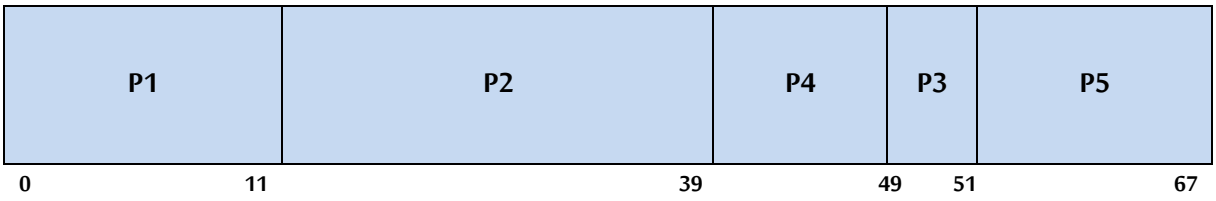
نبين فيما يلي مثالاً توضح آلية عمل هذه الخوارزمية

ليكن لدينا الجدول التالي الذين يبن أزمنة الوصول والتنفيذ والأولوية لخمس عمليات P1,P2,P3,P4,P5 :

Processes	Arrival Time	Burst Time	priority
P1	**0**	**11**	**2**
P2	**5**	**28**	**0**
P3	**12**	**2**	**3**
P4	**2**	**10**	**1**
P5	**9**	**16**	**4**

ملاحظة: باعتبار أن سير تنفيذ الخوارزمية يصبح مشابهاً لسير تنفيذ خوارزمية FCFS وذلك بعد القيام بترتيب العمليات حسب الأولوية وأزمنة الوصول، سنتجنب تكرار إيضاح آلية التنفيذ من خلال الجداول (كما قمنا مع الخوارزميتين السابقتين.

مخطط غانت Gantt لعملية التنفيذ السابقة :



تحقيق الخوارزمية بلغة C++

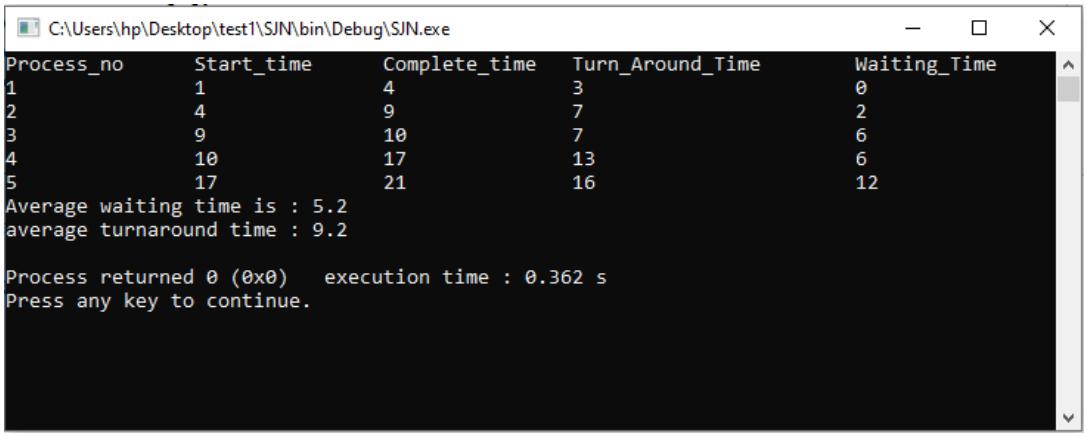
```
// C++ implementation for Priority Scheduling with Different Arrival Time
#include <bits/stdc++.h>
using namespace std;
#define totalprocess 5
// Making a struct to hold the given input
struct process
{
   int at,bt,pr,pno;
};
process proc[50];
// comparator function to sort according to priority

bool comp(process a,process b)
{
   if(a.at == b.at) {
        return a.pr<b.pr;
   }
   else{
           return a.at<b.at;
           }
}

// Using FCFS Algorithm to find Waiting time
void get_wt_time(int wt[])
{
   // declaring service array that stores cumulative burst time
```

```
  int service[50];
  // Initialising initial elements of the arrays
  service[0] = proc[0].at;
  wt[0]=0;
  for(int i=1;i<totalprocess;i++) {
       service[i]=proc[i-1].bt+service[i-1];
       wt[i]=service[i]-proc[i].at;
       // If waiting time is negative, change it into zero
       if(wt[i]<0) {
            wt[i]=0;
       }
       }
}

void get_tat_time(int tat[],int wt[])
{
  // Filling turnaroundtime array
  for(int i=0;i<totalprocess;i++) {
       tat[i]=proc[i].bt+wt[i];
  }
}

void findgc()
{
  //Declare waiting time and turnaround time array
  int wt[50],tat[50];
  double wavg=0,tavg=0;
  // Function call to find waiting time array
```

```
  get_wt_time(wt);
  //Function call to find turnaround time
  get_tat_time(tat,wt);
  int stime[50],ctime[50];
  stime[0] = proc[0].at;
  ctime[0]=stime[0]+tat[0];
  // calculating starting and ending time
  for(int i=1;i<totalprocess;i++)
        {
              stime[i]=ctime[i-1];
              ctime[i]=stime[i]+tat[i]-wt[i];
        }
cout<<"Process_no\tStart_time\tComplete_time\tTurn_Around_Time\tWaiting_Time"<<endl;

        // display the process details
        for(int i=0;i<totalprocess;i++)
        {
              wavg += wt[i];
              tavg += tat[i];
              cout<<proc[i].pno<<"\t\t"<<
                    stime[i]<<"\t\t"<<ctime[i]<<"\t\t"<< tat[i]<<"\t\t\t"<<wt[i]<<endl;
        }
        // display the average waiting time and average turn around time
        cout<<"Average waiting time is : ";
        cout<<wavg/(float)totalprocess<<endl;
        cout<<"average turnaround time : ";
        cout<<tavg/(float)totalprocess<<endl;
}
```

```
int main()
{
  int arrivaltime[] = { 1, 2, 3, 4, 5 };
  int bursttime[] = { 3, 5, 1, 7, 4 };
  int priority[] = { 3, 4, 1, 7, 8 };
        for(int i=0;i<totalprocess;i++) {
    proc[i].at=arrivaltime[i];
    proc[i].bt=bursttime[i];
    proc[i].pr=priority[i];
    proc[i].pno=i+1;
        }
        //Using inbuilt sort function
        sort(proc,proc+totalprocess,comp);
        //Calling function findgc for finding Gantt Chart
        findgc();
        return 0;
}
```

يبين الشكل ناتج تنفيذ هذا البرنامج:



خوارزمية الشريط الدوار Round Robin:

الشريط الدوار Round Robin عبارة عن خوارزمية جدولة لوحدة المعالجة المركزية حيث يتم تعيين فترة زمنية محددة لكل عملية بشكل دوري. إنها النسخة الوقائية من خوارزمية جدولة وحدة المعالجة المركزية (CPU) لمن يأتي أولاً يخدم أولاً.

- تركز خوارزمية Round Robin عمومًا على تقنية مشاركة الوقت.
- تسمى الفترة الزمنية التي يُسمح خلالها بتشغيل عملية أو وظيفة بطريقة وقائية الكم الزمني.
- يتم تعيين وحدة المعالجة المركزية (CPU) لكل عملية أو وظيفة موجودة في قائمة الانتظار الجاهزة، إذا اكتمل تنفيذ العملية خلال ذلك الوقت، فستنتهي العملية وإلا ستعود العملية إلى جدول الانتظار وتنتظر دورها التالي لاستكمال التنفيذ.

خصائص خوارزمية جدولة وحدة المعالجة المركزية Round Robin

- إنها بسيطة وسهلة التنفيذ ولا تتضمن إهمالاً لأي عملية حيث تحصل جميع العمليات على حصة عادلة من وحدة المعالجة المركزية.
- إحدى التقنيات الأكثر استخدامًا في جدولة وحدة المعالجة المركزية.
- هي خوارزمية استباقية حيث يتم تعيين العمليات لوحدة المعالجة المركزية (CPU) فقط لفترة زمنية ثابتة على الأكثر.
- عيبها الأساسي هو الحمل الزائد لتبديل السياق.

نبين فيما يلي مثالاً توضح آلية عمل هذه الخوارزمية

ليكن لدينا الجدول التالي الذين يبن أزمنة الوصول والتنفيذ لأربع عمليات P1,P2,P3,P4, بحيث أن الفاصل الزمني Time Quantum=2 :

Processes	Arrival Time	Burst Time
P1	**0 ms**	**5 ms**
P2	**1 ms**	**4 ms**
P3	**2 ms**	**2 ms**
P4	**4 ms**	**1 ms**

ستنفذ هذه الخوارزمية على خطوات كما يلي:

في اللحظة 0 :

يبدأ تنفيذ العملية P1 (التي زمن تنفيذها هو 5) حيث سيتم تنفيذ كل عملية لمدة 2 ميلي ثانية، وستكون P2 و P3 في جدول الانتظار.

Time Instance	Process	Arrival Time	Ready Queue	Running Queue	Execution Time	Initial Burst Time	Remaining Burst Time
0-2ms	**P1**	**0ms**	**P2, P3**	**P1**	**2ms**	**5ms**	**3ms**

في اللحظة 2 :

تنتقل العملية P1 إلى جدول الانتظار ويبدأ تنفيذ العملية P2 لمدة 2 ميلي ثانية.

Time Instance	Process	Arrival Time	Ready Queue	Running Queue	Execution Time	Initial Burst Time	Remaining Burst Time
2-4ms	**P1**	**0ms**	**P3, P1**	**P2**	**0ms**	**3ms**	**3ms**
	P2	**1ms**			**2ms**	**4ms**	**2ms**

في اللحظة 4:

تنتقل العملية P2 إلى جدول الانتظار ويبدأ تنفيذ العملية P3 لمدة 2 ميلي ثانية.

Time Instance	Process	Arrival Time	Ready Queue	Running Queue	Execution Time	Initial Burst Time	Remaining Burst Time
4-6ms	P1	0ms	P1, P4, P2	P3	0ms	3ms	3ms
	P2	1ms			0ms	2ms	2ms
	P3	2ms			2ms	2ms	0ms

في اللحظة 6:

ينتهي تنفيذ العملية P3 وتعود العملية P1 إلى متابعة التنفيذ

Time Instance	Process	Arrival Time	Ready Queue	Running Queue	Execution Time	Initial Burst Time	Remaining Burst Time
6-8ms	P1	0ms	P4, P2	P1	2ms	3ms	1ms
	P2	1ms			0ms	2ms	2ms

في اللحظة 8:

يبدأ تنفيذ العملية P4 وباعتبار زمن تنفيذها هو 1 ميلي ثانية بالتالي سوف لن يستغرق التنفيذ سوى 1 ميلي ثانية.

Time Instance	Process	Arrival Time	Ready Queue	Running Queue	Execution Time	Initial Burst Time	Remaining Burst Time
8-9ms	P1	0ms	P2, P1	P4	0ms	3ms	1ms
	P2	1ms			0ms	2ms	2ms
	P4	4ms			1ms	1ms	0ms

في اللحظة 9:

تنهي العملية P4 تنفيذها وتعود العملية P2 لاستكمال تنفيذها.

Time Instance	Process	Arrival Time	Ready Queue	Running Queue	Execution Time	Initial Burst Time	Remaining Burst Time
9-11ms	P1	0ms	P1	P2	0ms	3ms	1ms
	P2	1ms			2ms	2ms	0ms

في اللحظة 11:

تنهي العملية P2 تنفيذها وتعود العملية P1 لاستكمال تنفيذها لمدة 1 ميلي ثانية فقط.

Time Instance	Process	Arrival Time	Ready Queue	Running Queue	Execution Time	Initial Burst Time	Remaining Burst Time
11-12ms	P1	0ms		P1	1ms	1ms	0ms

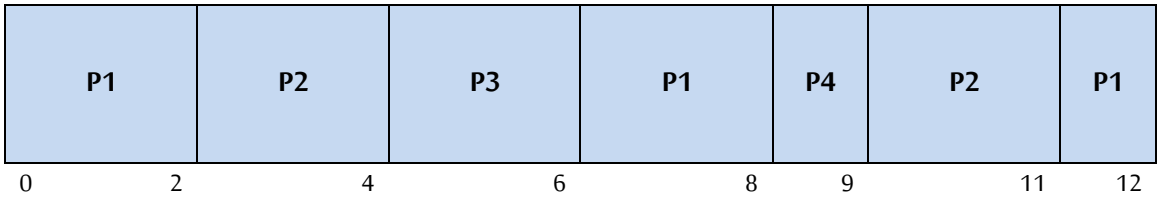
في اللحظة 12:

تنهي العملية P1 تنفيذها وبالتالي يكون الجدول الكلي الذي يلخص التنفيذ على النحو:

Time Instance	Process	Arrival Time	Ready Queue	Running Queue	Execution Time	Initial Burst Time	Remaining Burst Time
0-2ms	P1	0ms	P2, P3	P1	2ms	5ms	3ms
2-4ms	P1	0ms	P3, P1	P2	0ms	3ms	3ms
	P2	1ms			2ms	4ms	2ms
4-6ms	P1	0ms	P1, P4, P2	P3	0ms	3ms	3ms
	P2	1ms			0ms	2ms	2ms
	P3	2ms			2ms	2ms	0ms
6-8ms	P1	0ms	P4, P2	P1	2ms	3ms	1ms
	P2	1ms			0ms	2ms	2ms

8-9ms	P1	0ms	P2, P1	P4	0ms	3ms	1ms
	P2	1ms			0ms	2ms	2ms
	P4	4ms			1ms	1ms	0ms
9-11ms	P1	0ms	P1	P2	0ms	3ms	1ms
	P2	1ms			2ms	2ms	0ms
11-12ms	P1	0ms		P1	1ms	1ms	0ms

مخطط غانت Gantt لعملية التنفيذ السابقة :



الحسابات الزمنية في خوارزمية الشريط الدوار:

- زمن الانتهاء Completion Time وهو الزمن الذي تنهي فيه كل عملية عملها.
- زمن الدوران Turn Around Time وهو الفرق بين زمن الانتهاء وزمن الوصول.
- زمن الانتظار Waiting Time وهو الفرق بين زمن الدوران وزمن التنفيذ.

Processes	AT	BT	CT	TAT	WT
P1	0	5	12	12-0 = 12	12-5 = 7
P2	1	4	11	11-1 = 10	10-4 = 6
P3	2	2	6	6-2 = 4	4-2 = 2
P4	4	1	9	9-4 = 5	5-1 = 4

Average Turn around time = (12 + 10 + 4 + 5)/4 = 31/4 = 7.7

Average waiting time = (7 + 6 + 2 + 4)/4 = 19/4 = 4.7

تحقيق الخوارزمية بلغة C++

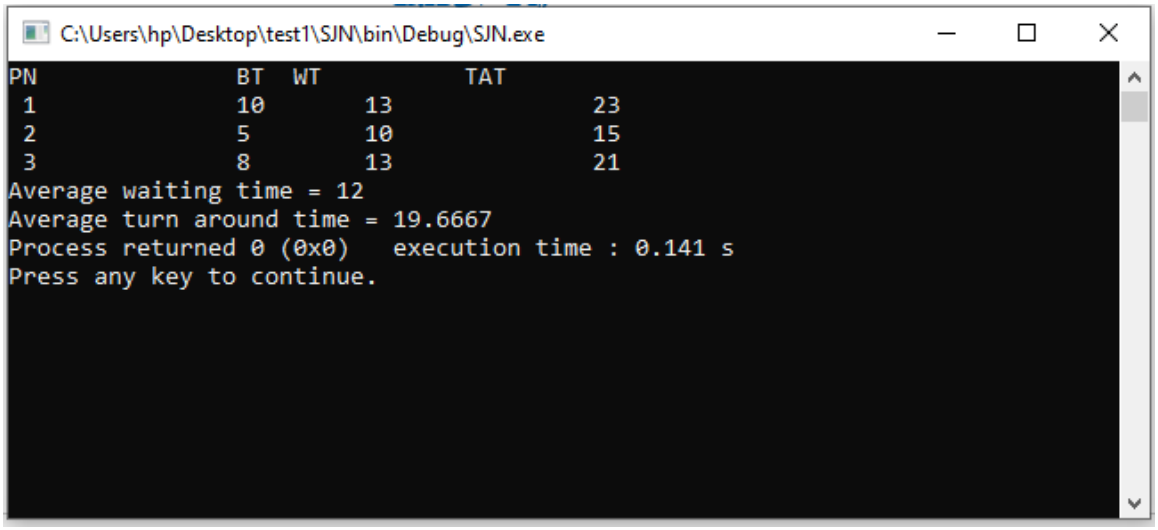
```
// C++ program for implementation of RR scheduling
#include<iostream>
using namespace std;
// Function to find the waiting time for all processes
void findWaitingTime(int processes[], int n, int bt[], int wt[], int quantum)
{
	// Make a copy of burst times bt[] to store remaining burst times.
	int rem_bt[n];
	for (int i = 0 ; i < n ; i++)
		rem_bt[i] = bt[i];
	int t = 0; // Current time
	// Keep traversing processes in round robin manner until all of them are not done.
	while (1)
	{
		bool done = true;
		// Traverse all processes one by one repeatedly
		for (int i = 0 ; i < n; i++)
		{
		// If burst time of a process is greater than 0 then only need to process further
			if (rem_bt[i] > 0)
			{
				done = false; // There is a pending process
				if (rem_bt[i] > quantum)
				{
	// Increase the value of t i.e. shows  how much time a process has been processed
```

```
                    t += quantum;
            // Decrease the burst_time of current process by quantum
                    rem_bt[i] -= quantum;
                }
        // If burst time is smaller than or equal to quantum. Last cycle for this process
                else
                {
        // Increase the value of t i.e. shows how much time a process has been processed
                    t = t + rem_bt[i];
                // Waiting time is current time minus time used by this process
                    wt[i] = t - bt[i];
            // As the process gets fully executed make its remaining burst time = 0
                    rem_bt[i] = 0;
                }
            }
            }
            // If all processes are done
            if (done == true)
            break;
        }
}

// Function to calculate turn around time
void findTurnAroundTime(int processes[], int n,int bt[], int wt[], int tat[])
{
        // calculating turnaround time by adding bt[i] + wt[i]
        for (int i = 0; i < n ; i++)
            tat[i] = bt[i] + wt[i];
```

```
}

// Function to calculate average time
void findavgTime(int processes[], int n, int bt[],int quantum)
{
        int wt[n], tat[n], total_wt = 0, total_tat = 0;
        // Function to find waiting time of all processes
        findWaitingTime(processes, n, bt, wt, quantum);
        // Function to find turn around time for all processes
        findTurnAroundTime(processes, n, bt, wt, tat);
        // Display processes along with all details
        cout << "PN\t "<< " \tBT " << " WT " << " \tTAT\n";

        // Calculate total waiting time and total turn around time
        for (int i=0; i<n; i++)
        {
                total_wt = total_wt + wt[i];
                total_tat = total_tat + tat[i];
                cout << " " << i+1 << "\t\t" << bt[i] <<"\t " << wt[i] <<"\t\t " << tat[i] <<endl;
        }
        cout << "Average waiting time = " << (float)total_wt / (float)n;
        cout << "\nAverage turn around time = " << (float)total_tat / (float)n;
}

// Driver code
int main()
{
        int processes[] = { 1, 2, 3};                        // process id's
```

```
        int n = sizeof processes / sizeof processes[0];
        int burst_time[] = {10, 5, 8};                  // Burst time of all processes
        int quantum = 2;                                 // Time quantum
        findavgTime(processes, n, burst_time, quantum);
        return 0;
}
```

يكون ناتج تنفيذ البرنامج



نهاية الجلسة