

## جامعة المنارة

كلية: ..... الهندسة.....

قسم: ..... الهندسة المعلوماتية.....

اسم المقرر: ..... نظم تشغيل 2.....

رقم الجلسة (2....)

عنوان الجلسة

خوارزميات إدارة العمليات متعددة المستويات وجدولتها

م.فاطمة جنيدى

م.عمار مصطفى



العام الدراسي

2025/2024

الفصل الدراسي

## جدول المحتويات

### Contents

رقم الصفحة	العنوان
1	جدولة طوابير الانتظار متعددة المستويات
8	جدولة الطوابير ذو التغذية الراجعة متعددة المستويات

**الغاية من الجلسة:** تعريف الطالب بخوارزميات توزيع العمليات ضمن طابور الانتظار إلى عدة مستويات حسب الأولوية المرتبطة بنوع العملية أو طوابير ديناميكية تتغير حسب زمن التنفيذ

## جدولة طوابير الانتظار متعددة المستويات

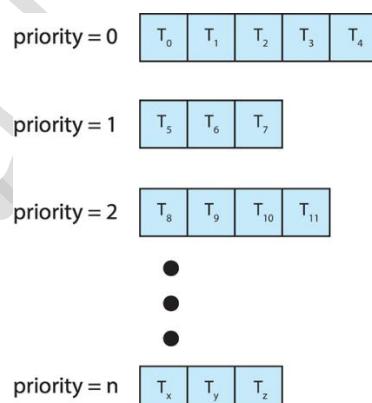
### Multilevel Queue Scheduling

يتكون طابور الانتظار الجاهزة من عدة طوابير انتظار

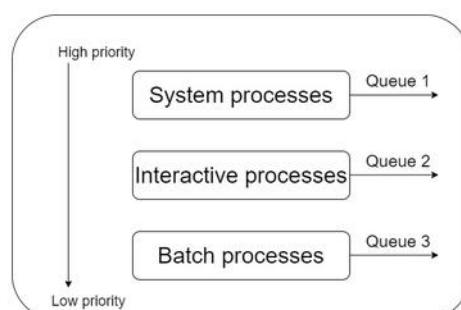
جدول طابور الانتظار متعدد المستويات يتم تحديده بالمعلمات التالية:

- عدد طوابير الانتظار
- خوارزميات الجدولة لكل طابور انتظار
- الطريقة المستخدمة لتحديد طابور الانتظار التي ستدخلها العملية عندما تحتاج هذه العملية إلى الخدمة
- الجدولة بين طوابير الانتظار

مع جدولة الأولويات، يمكننا تخصيص طوابير منفصلة لكل أولوية: نقوم بجدولة العملية في طابور الانتظار ذات الأولوية الأعلى!



تحديد الأولويات بناءً على نوع العملية



عندما تأتي عملية جديدة، تُضاف هذه العملية إلى واحدة من طوابير العمليات الثلاث المذكورة أعلاه بناءً على التصنيف المحدد لطوابير العمليات هذه. لنفترض أن عملية تفاعلية مثل الألعاب عبر الإنترنت تريد استخدام وحدة المعالجة المركزية.

بعد ذلك، سيتم وضع هذه العملية في طابور العمليات التفاعلية. أو إذا جاءت أي عملية مملوكة لنظام التشغيل نفسه، يتم وضعها في طابور عمليات النظام، وعلى نحو مماثل بالنسبة للعمليات الأخرى.

الآن لدينا السؤال حول أي عملية ستذهب أولاً إلى وحدة المعالجة المركزية إذا كانت جميع طوابير العمليات تحتوي على بعض العمليات. يمكن حل هذه المعضلة بطرق مختلفة. لقد ناقشنا طريقتين مختلفتين لحلها.

#### 1. طريقة الدولة الاستباقية ذات الأولوية الثانية

يتم تعين أولوية مطلقة لكل طابور انتظار على طوابير الانتظار الأخرى في هذه الطريقة. على سبيل المثال، ليكن لدينا ترتيب الأولوية التالي:

طابور انتظار عمليات النظام < طابور انتظار العمليات التفاعلية > طابور انتظار العمليات الدفعية

وفقاً لترتيب الأولوية هذا، يتم تشغيل جميع العمليات في طابور انتظار عمليات النظام في البداية، ولا ينتقل إلى طابور انتظار العمليات التالية حتى تصبح طابور انتظار العمليات الحالية فارغة. عندما يتم تنفيذ جميع عمليات طابور انتظار النظام، ينتقل إلى طابور انتظار العمليات ذات الأولوية التالية. وعلى نحو مماثل، بالنسبة لطوابير الانتظار ذات الأولوية الأقل التالية.

لنفترض أنتا تقوم بتشغيل العمليات من طابور انتظار العمليات التفاعلية، وتصل عملية جديدة إلى طابور انتظار عمليات النظام ذات الأولوية الأعلى. عندئذٍ سنقوم بتشغيل عملية النظام أولاً، ثم ينتقل إلى العملية التفاعلية، وهذا فقط عندما تكون طابور انتظار عمليات النظام فارغة. لذا فإن جوهر هذه الطريقة هو تشغيل العمليات ذات الأولوية الأعلى أولاً ثم العمليات ذات الأولوية اللاحقة.

#### 2. طريقة تقسيم الوقت Time slicing

في هذه الطريقة يتم تشغيل العمليات من كل طابور انتظار لمدة زمنية محددة، ثم ينتقل إلى طابور انتظار التالية. وعندما نصل إلى طابور الانتظار الأخيرة، نقوم بتشغيلها لمدة زمنية محددة، ثم نعود إلى طابور الانتظار الأولى.

على سبيل المثال، يتم تشغيل طابور الانتظار 1 لمدة 50% من وقت وحدة المعالجة المركزية، ويتم تشغيل طابور الانتظار الثانية لمدة 30% من وقت وحدة المعالجة المركزية، ويتم تشغيل طابور الانتظار الثالثة لمدة 20% من وقت وحدة المعالجة المركزية. وهذه الطريقة، يتم تقسيم إجمالي وقت وحدة المعالجة المركزية بين كل طابور انتظار لتشغيل عملياتها.

لقد حللنا مشكلة العمليات التي سيتم تشغيل طابور الانتظار الخاصة بها أولاً. والآن، نأتي إلى مصير طوابير الانتظار الفردية. في كل طابور انتظار، لدينا عمليات متعددة، فكيف نقرأ أي عملية يجب تشغيلها أولاً ولأي وقت. وهل تحتاج إلى تشغيل العمليات في جميع طوابير الانتظار بنفس الطريقة التي قمنا بها في طابور الانتظار الأولى، أم يمكننا تنفيذ طرق مختلفة لتشغيل العمليات في طوابير انتظار مختلفة؟

الإجابة على هذا السؤال هي أنه يمكننا أن نمتلك تطبيقات مختلفة لكل طابور انتظار. وهناك العديد من الخوارزميات التي يمكننا من خلالها تحديد العملية التي سيتم تشغيلها أولاً ومتى تشغيلها.

على سبيل المثال:

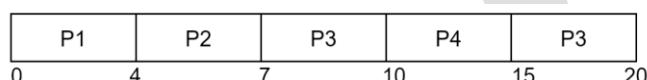
- يمكن لطابور انتظار عمليات النظام استخدام جدولة FCFS (أول من يأتي، أول من يخدم).
- يمكن لطابور انتظار العمليات التفاعلية استخدام جدولة SJF (أقصر وظيفة أولًا).
- يمكن لطابور انتظار العمليات الدفعية استخدام جدولة RR (دوره روبن).

فيما يلي مثال لفهم العمل الإجمالي لخوارزمية جدولة طابور الانتظار متعددة المستويات.

لنأخذ بعض الاعتبار العمليات الأربع التالية:

Process	Burst time	Arrival time	Queue number
P1	4	0	1
P2	3	0	2
P3	8	0	3
P4	5	10	1

لنفترض أن ترتيب أولوية الطوابير هو كما يلي: Queue1 > Queue2 > Queue3 ، سينبدي مخطط جانت كما يلي:



في هذا المثال، تصل العمليات P1 و P2 و P3 إلى  $t=0$ ، ولكن مع ذلك، تعمل P1 أولاً لأنها تنتهي إلى طابور الانتظار رقم 1، والتي لها أولوية أعلى. بعد انتهاء عملية P1، تعمل عملية P2 في طابور الانتظار 2 قبل العملية P3 الموجودة في طابور الانتظار 3 لأن طابور الانتظار 2 لها أولوية أعلى من طابور الانتظار 3، ثم تعمل P3. أثناء تشغيل عملية P3، تصل العملية P4 التي تنتهي إلى طابور الانتظار 1. نظرًا لأن طابور الانتظار 1 لها أولوية أعلى من طابور الانتظار 3، يتم إيقاف العملية P3 (إيقافها مؤقتًا)، وتبدأ P4 في التنفيذ. بعد اكتمال تنفيذ P4، يتم استئناف تنفيذ P3.

تطبيق C++ لجدولة طوابير الانتظار متعددة المستويات:

```

#include <stdio.h> // For input/output functions like printf and scanf
#include <stdlib.h> // For system("pause")

int main() {
    int p[20], bt[20], su[20], wt[20], tat[20], i, k, n, temp;
    float wtavg, tatavg;
}

```

[20]p: مصفوفة لتخزين معرفات العملية (بافتراض وجود 20 عملية كحد أقصى).

[20]bt: مصفوفة لتخزين زمن الرشقة (وقت تنفيذ وحدة المعالجة المركزية) لكل عملية.

[20]su: مصفوفة لتخزين ما إذا كانت العملية عملية نظام (0) أو عملية مستخدم (1).

[20]wt: مصفوفة لتخزين وقت الانتظار لكل عملية.

[20]: مصفوفة لتخزين وقت الدوران لكل عملية.

i, k, n: عدادات الحلقة وعدد العمليات.

temp: متغير مؤقت يستخدم للتبدل أناء الفرز.

wtavg, tatavg: متغيرات لتخزين متوسط وقت الانتظار ومتوسط وقت الدوران.

```
printf("Enter the number of processes:");
scanf("%d", &n);

for (i = 0; i < n; i++) {
    p[i] = i; // Assign process IDs
    printf("Enter the Burst Time of Process %d:", i);
    scanf("%d", &bt[i]);
    printf("System/User Process (0/1):");
    scanf("%d", &su[i]);
}
```

يطلب الكود بإدخال عدد العمليات، ثم يأخذ في حلقة مفرغة رشقة المعالجة ونوع عملية النظام/المستخدم لكل عملية كمدخلات. يتم تعيين معرفات العمليات تلقائياً.

```
for (i = 0; i < n; i++)
    for (k = i + 1; k < n; k++)
        if (su[i] > su[k]) {
            // Swap process information if su[i] > su[k] (system processes have priority)
            temp = p[i];
            p[i] = p[k];
            p[k] = temp;
            temp = bt[i];
            bt[i] = bt[k];
            bt[k] = temp;
            temp = su[i];
            su[i] = su[k];
            su[k] = temp;
        }
```

يتم هنا فرز العمليات بناءً على مجموعة su. يتم إعطاء الأولوية لعمليات النظام (0) على عمليات المستخدم (1). يتم الفرز باستخدام خوارزمية فرز الفقاعات. لاحظ أن هذه خوارزمية فرز غير فعالة للغاية لمجموعات البيانات الأكبر.

```
wtavg = wt[0] = 0; // The first process has no waiting time
tatavg = tat[0] = bt[0]; // Turnaround time for the first process is its burst time

for (i = 1; i < n; i++) {
    wt[i] = wt[i - 1] + bt[i - 1]; // Waiting time = previous process's waiting time + previous process's burst time
    tat[i] = tat[i - 1] + bt[i]; // Turnaround time = previous process's turnaround time + current process's burst time
    wtavg = wtavg + wt[i];
    tatavg = tatavg + tat[i];
}
```

يتم هنا حساب زمن الانتظار وقت التنفيذ لكل عملية باستخدام خوارزمية FCFS. زمن انتظار العملية هو مجموع أزمنة الرشقات لجميع العمليات السابقة. زمن التنفيذ هو مجموع زمن الانتظار وזמן الرشقة

```

printf("\nPROCESS\tSYSTEM/USER PROCESS\tBURST TIME\tWAITING TIME\tTURNAROUND TIME");
for (i = 0; i < n; i++)
    printf("\n%d\t%d\t%d\t%d\t%d", p[i], su[i], bt[i], wt[i], tat[i]);
printf("\nAverage Waiting Time is --- %f", wtavg / n);
printf("\nAverage Turnaround Time is --- %f", tatavg / n);
system("pause"); // Pause the console to see the output
return 0;
}

```

أخيراً، يقوم الكود بطباعة جدول يوضح معرف العملية ونوع النظام/المستخدم وزمن الرشقة، وزمن الانتظار وزمن التنفيذ لكل عملية. كما يحسب ويعرض متوسط زمن الانتظار ومتوسط زمن التنفيذ. ("pause") هي دالة غير قياسية تبقى نافذة وحدة التحكم مفتوحة بعد انهاء البرنامج من التنفيذ، مما يسمح للمستخدم برؤيه الناتج. هذه الدالة غير قابلة للنقل ويجب تجنبها في الكود الإنتاجي. سيكون النهج الأفضل هو استخدام طريقة خاصة بالمنصة لإيقاف التنفيذ مؤقتاً أو ببساطة إزالة السطر.

```

Enter the number of processes:3
Enter the Burst Time of Process0:12
System/User Process (0/1) ? 0
Enter the Burst Time of Process1:18
System/User Process (0/1) ? 0
Enter the Burst Time of Process2:15
System/User Process (0/1) ? 1

      PROCESS      SYSTEM/USER PROCESS      BURST TIME      WAITING TIME      TURNAROUND TIME
0          0                  12                  0                  12
1          0                  18                  12                 30
2          1                  15                 30                 45

Average Waiting Time is --- 14.000000
Average Turnaround Time is --- 29.000000

```

هل جدولة طابور الانتظار متعددة المستويات استباقية أم غير استباقية؟

يمكن أن تكون جدولة طابور الانتظار متعددة المستويات استباقية أم غير استباقية بطبيعتها، اعتماداً على خوارزمية الجدولة المستخدمة في كل طابور انتظار. على سبيل المثال، تكون خوارزمية Round Robin استباقية، في حين أن خوارزمية First Come, First Serve غير استباقية.

ما هي أنواع العمليات التي لها أعلى أولوية في جدولة طابور الانتظار متعددة المستويات؟

يتم إعطاء الأولوية القصوى للعمليات التفاعلية لأنها عمليات تتفاعل مع المستخدم، مثل متصفحات الويب ومعالجات الكلمات وعملاء البريد الإلكتروني. وذلك حتى يحصل المستخدمون على الاستجابة بسرعة.

ما الذي تعاني منه جدولة طابور الانتظار متعددة المستويات؟

يمكن أن تعاني جدولة طابور الانتظار متعددة المستويات من التكرار. يحدث التكرار عندما يتم التكرار باستمرار بواسطة عمليات أخرى ذات أولوية أعلى. وهذا يقلل من أداء النظام.

حالة التجويع للعمليات ذات المستوى الأدنى، بسبب التجويع، إما أن العمليات ذات المستوى الأدنى لا يتم تنفيذها مطلقاً أو تضطر إلى الانتظار لفترة طويلة من الوقت بسبب الأولوية المنخفضة أو أن العملية ذات الأولوية الأعلى تستغرق قدرًا كبيرًا من الوقت.

هل جدولة FIFO استباقية أم غير استباقية؟

جدولة FIFO (الأول في الدخول، الأول في الخروج) هي خوارزمية جدولة غير استباقية، حيث يتم تنفيذ العملية التي تصل أولاً أولاً وتستمر حتى الانتهاء قبل أن تبدأ العملية التالية، دون أي انقطاع.

## جدولة الطوابير ذو التغذية الراجعة متعددة المستويات

### Multilevel Feedback Queue

تعد جدولة طوابير ذو التغذية الراجعة متعددة المستويات (MLFQ) طريقة متقدمة لجدولة وحدة المعالجة المركزية تعتمد على نهج طابور انتظار متعددة المستويات (MLQ)، ولكن مع إمكانية إضافية لتمكين العمليات من التنقل بين طوابير الانتظار. تعمل هذه المرونة على تعزيز الكفاءة مقارنة بجدولة طابور انتظار متعددة المستويات التقليدية.

يتم إنشاء خوارزميات الجدولة لإبقاء وحدة المعالجة المركزية (CPU) مشغولة. يتم توسيع الخوارزميات القياسية بواسطة طابور التغذية الراجعة متعدد الطبقات، والذي يحتوي على متطلبات التصميم التالية:

- تقسيم العمليات إلى طوابير جاهزة مختلفة بناءً على متطلبات المعالج.
- إعطاء الأولوية للعمليات ذات دفعات وحدة المعالجة المركزية القصيرة.
- إعطاء الأولوية للعمليات ذات دفعات الإدخال/الإخراج العالية.

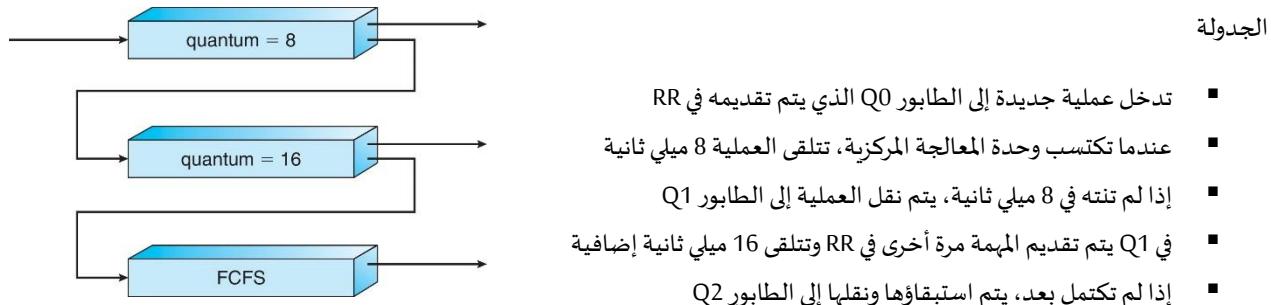
يقوم MLFQ (جدولة طابور التغذية الراجعة متعددة المستويات) بتحليل سلوك العمليات (وقت التنفيذ) بشكل مستمر وضبط أولويتها وفقاً لذلك.

جدولة طوابير انتظار المرونة متعددة المستويات هي خوارزمية جدولة وحدة المعالجة المركزية التي تقوم بتعيين العمليات إلى طوابير انتظار مختلفة بناءً على أولويتها وتاريخ استخدام الموارد. تستخدم الخوارزمية طوابير انتظار متعددة بأولويات مختلفة وكمية زمنية مختلفة، ويتم نقل العمليات بين الطوابير بناءً على سلوكها وأدائها، مما يحسن كفاءة النظام بشكل عام.

يتم تحديد جدولة طابور انتظار ردود الفعل متعددة المستويات من خلال المعلومات التالية:

- عدد الطوابير
- خوارزميات الجدولة لكل طابور انتظار
- الطريقة المستخدمة لتحديد وقت ترقية عملية
- الطريقة المستخدمة لتحديد وقت خفض مرتبة عملية
- الطريقة المستخدمة لتحديد طابور الانتظار التي ستتدخلها عملية ما عندما تحتاج هذه العملية إلى خدمة
- يمكن تنفيذ التقادم باستخدام طابور انتظار ردود الفعل متعددة المستويات

مثال: ليكن لدينا ثلاثة طوابير: RR – Q0 مع وقت كمي 8 ملي ثانية، Q1 – RR و Q2 – FCFS مع وقت كمي 16 ملي ثانية.



#### ميزات جدولة طو ايرذو التغذية الراجعة متعددة المستويات (MLFQ):

- مستويات أولوية متعددة MLFQ: تنظم العمليات في طوابير أولوية متعددة، كل منها يمثل مستويات مختلفة من الأولوية بناءً على خصائص العملية مثل وقت اندفاع وحدة المعالجة المركزية أو السلوك التاريخي.
- تعيين الأولوية الديناميكي Dynamic Priority Assignment: تنتقل العمليات بين طوابير الأولوية بشكل ديناميكي بناءً على سلوكها واستخدام الموارد. قد تصاعد العمليات ذات الأولوية المنخفضة إلى طوابير أولوية أعلى إذا أظهرت سمات سلوك مرتبط بوحدة المعالجة المركزية على المدى القصير.
- تباين شريحة الوقت Time Slice Variation: يتم تخصيص كمية زمنية مختلفة أو شريحة زمنية لكل طابور أولوية. تتنقل طوابير الأولوية الأعلى عادةً شريحة زمنية أقصر لتعزيز الاستجابة، بينما تتنقل طوابير الأولوية المنخفضة شريحة زمنية أطول لمنع الماجعة.
- الاستبدال والنضوج Preemption and Aging: يستخدم MLFQ الاستبدال لضمان قدرة العمليات ذات الأولوية العالية على مقاطعة العمليات ذات الأولوية المنخفضة إذا لزم الأمر. بالإضافة إلى ذلك، تتنقل العمليات التي تنتظر في طوابير ذات أولوية أقل تدريجياً أولوية متزايدة بمرور الوقت لمنع الحظر غير المحدد أو التجويع.
- الجدولة التكيفية Adaptive Scheduling: تضبط MLFQ معلمات الجدولة الخاصة بها ديناميكياً بناءً على حمل النظام وسلوك العملية. وتهدف إلى إيجاد توازن بين العدالة والاستجابة والكافأة في استخدام الموارد.
- تجنب عكس الأولوية Avoidance of Priority Inversion: يساعد MLFQ في منع مشكلات عكس الأولوية من خلال السماح للعمليات ذات الأولوية الأعلى باستباق العمليات ذات الأولوية المنخفضة، مما يضمن تنفيذ المهام الحرجة في الوقت المناسب.

#### الحاجة إلى جدولة الطو ايرذو التغذية الراجعة متعددة المستويات

هذه الطريقة أكثر قابلية للتكييف من جدولة طوابير انتظار المرنة متعددة المستويات. تساهم هذه الطريقة في تسريع وقت الاستجابة.

طريقة SJF، التي تتطلب مدة تشغيل العمليات لجدولتها، مطلوبة لتحسين وقت الاستجابة. وكما نعلم جميعاً، لا يمكن التنبؤ بمسار الإجراء مسبقاً. وعلاوة على ذلك، فإن هذا الجدولية يقوم في الغالب بتشغيل عملية لفترة زمنية محددة قبل تغيير أولوية العملية إذا كانت العملية طويلة. ونتيجة لذلك، تتعلم خوارزمية الجدولة الخاصة بما في المقام الأول من سلوك العملية السابقة قبل التنبؤ بسلوك العملية المستقبلية. ونتيجة لذلك، تسعى MFQS إلى تنفيذ عملية أقصر أولاً، مما يساعد على تقليل وقت الاستجابة.

بداية، يتم إكمال جميع العمليات في Q1. يتم تشغيل العمليات في Q2 عندما يكون Q1 فارغاً. عندما يكون Q1 و Q2 فارغين، سيتم تشغيل عمليات Q3. إذا تم جدولة إجراء ما في Q2، فسوف يكون له الأولوية على العملية المخطط لها في Q3. وبالمثل، إذا وصلت عملية إلى Q3 قبل العملية الحالية، فسوف تسبقها في Q3. في Q3، يتم إجراء العمليات على أساس FCFS.

#### تنفيذ MFQS

عندما تبدأ عملية ما في التشغيل، يتم وضعها في طابور الانتظار 1. لا تغير أولوية عملية طابور الانتظار 1 سواء كانت تكتمل أو تعطى وحدة المعالجة المركزية لعملية الإدخال/الإخراج كم زمني 4 ميلي ثانية، وإذا عادت إلى طابور الانتظار الجاهزة، فإنها تستأنف تنفيذها في طابور الانتظار 1.

إذا لم تنته عملية في طابور الانتظار 1 في كم زمني 4 ميلي ثانية، فإنها تفقد الأولوية وستنتقل إلى طابور الانتظار 2. تتطبق النقاطان 2 و 3 على عمليات طابور الانتظار 2 أيضاً. ومع ذلك، فإن الكم الزمني هو ثمانى وحدات. إذا لم تنته عملية في كم زمني، فسوف تنتقل إلى الكم الزمني التالي.

تم جدولة العمليات بطريقة FCFS في طابور الانتظار المائي. لا يمكن تشغيل عمليات طابور الانتظار ذات الأولوية المنخفضة إلا عندما تكون طوابير الانتظار ذات الأولوية الأعلى فارغة. يتم إيقاف عملية طابور الانتظار ذات الأولوية المنخفضة مؤقتاً عند وصول وظيفة طابور انتظار ذات أولوية أعلى

#### تطبيق عملي:

قم بتصميم نواة نظام تشغيل صغيرة تستخدم جدولة الطوایر ذو التغذية الراجعة متعددة المستويات وفق ما يلي:

ليكن لدينا ثلاثة طوابير كما هو موضح في الجدول التالي:

أولوية عالية	الشريحة الزمنية = 2	خوارزمية Round-Robin	Q1
	الشريحة الزمنية = 4	خوارزمية Round-Robin	Q2
أولوية منخفضة		خوارزمية SJF غير استباقيه	Q3

- جميع العمليات الجديدة تدخل الطابور Q1
- في الطابور Q1 و Q2 إذا لم تنتهي العملية ضمن الشريحة الزمنية الخاصة بها سيتم نقلها إلى الطابور ذو الأولوية المنخفضة التالي.
- إذا كان زمن الانتظار للعملية أكبر أو يساوي 13، يتم ترقية العملية إلى الطابور ذو الأولوية الأعلى التالي.

Process	Arrival Time	Burst Time
P1	3	8
P2	4	7
P3	8	7
P4	12	1

	Q1(2)	Q2(4)	Q3(SJF)	W(P1)	W(P2)	W(P3)	W(P4)
3	P1(8)						
4	P1(7),P2(7)						
5	P2(7)	P1(6)			1		
7		P1(6),P2(5)		2	1		
8	P3(7)	P1(5),P2(5)					
11	P3(7)	P2(5)	P1(2)	2	5	3	
12	P3(6),P4(1)	P2(5)	P1(2)				
13	P4(1)	P2(5),P3(5)	P1(2)	4	7	3	1
14		P2(5),P3(5)	P1(2)	5	8	4	
18		P3(5)	P1(2),P2(1)	9	8	8	
22		P1(2)	P2(1),P3(1)	13	12	8	
24		P2(1)	P3(1)		14	10	
25			P3(1)		11		

### تطبيق C++ لجدولة طوابير الانتظار المرنة متعددة المستويات:

```

1 #include <stdio.h>
2
3 struct process
4 {
5     char name;
6     int AT, BT, WT, TAT, RT, CT;
7 } Q1[10], Q2[10], Q3[10]; /*Three queues*/
8
9 int n;
10 void sortByArrival()
11 {
12     struct process temp;
13     int i, j;
14     for (i = 0; i < n; i++)
15     {
16         for (j = i + 1; j < n; j++)
17         {
18             if (Q1[i].AT > Q1[j].AT)
19             {
20                 temp = Q1[i];
21                 Q1[i] = Q1[j];
22                 Q1[j] = temp;
23             }
24         }
25     }
26 }
27
28 int main()
29 {
30     int i, j, k = 0, r = 0, time = 0, tq1 = 8, tq2 = 16, flag = 0;
31     char c;
32     int sumW = 0;
33     int sumT = 0;
34     int sumR = 0;

```

```

35     printf("Enter no of processes:");
36     scanf("%d", &n);
37     for (i = 0, c = 'A'; i < n; i++, c++)
38     {
39         Q1[i].name = c;
40         printf("\nEnter the arrival time and burst time of process %c: ", Q1[i].name);
41         scanf("%d%d", &Q1[i].AT, &Q1[i].BT);
42         Q1[i].RT = Q1[i].BT; /*save burst time in remaining time for each process*/
43     }
44     sortByArrival();
45     time = Q1[0].AT;
46     printf("Process in first queue following RR with qt=8");
47     printf("\nProcess\t\tRT\t\tWT\t\tTAT\t\t");
48     for (i = 0; i < n; i++)
49     {
50
51         if (Q1[i].RT <= tq1)
52         {
53
54             time += Q1[i].RT; /*from arrival time of first process to completion of this
process*/
55             Q1[i].RT = 0;
56             Q1[i].WT = time - Q1[i].AT - Q1[i].BT; /*amount of time process has been
waiting in the first queue*/
57             Q1[i].TAT = time - Q1[i].AT; /*amount of time to execute the
process*/
58             printf("\n%c\t\t%d\t\t%d\t\t%d", Q1[i].name, Q1[i].BT, Q1[i].WT, Q1[i].TAT);
59             sumW += Q1[i].WT;
60             sumT += Q1[i].TAT;
61             sumR += Q1[i].BT;
62         }
63         else /*process moves to queue 2 with qt=8*/
64         {
65             Q2[k].WT = time;
66             time += tq1;
67             Q1[i].RT -= tq1;
68             Q2[k].BT = Q1[i].RT;
69             Q2[k].RT = Q2[k].BT;
70             Q2[k].name = Q1[i].name;
71             k = k + 1;
72             flag = 1;
73         }
74     }
75     if (flag == 1)
76     {
77         printf("\nProcess in second queue following RR with qt=16");
78         printf("\nProcess\t\tRT\t\tWT\t\tTAT\t\t");
79     }
80     for (i = 0; i < k; i++)
81     {
82         if (Q2[i].RT <= tq2)
83         {
84             time += Q2[i].RT; /*from arrival time of first process +BT of this process*/
85             Q2[i].RT = 0;
86             Q2[i].WT = time - tq1 - Q2[i].BT; /*amount of time process has been waiting
in the ready queue*/
87             Q2[i].TAT = time - Q2[i].AT; /*amount of time to execute the process*/
88             printf("\n%c\t\t%d\t\t%d\t\t%d", Q2[i].name, Q2[i].BT, Q2[i].WT, Q2[i].TAT);
89             sumW += Q2[i].WT;
90             sumT += Q2[i].TAT;
91             sumR += Q2[i].BT;
92         }
93         else /*process moves to queue 3 with FCFS*/
94         {
95             Q3[r].AT = time;
96             time += tq2;
97             Q2[i].RT -= tq2;
98             Q3[r].BT = Q2[i].RT;
99             Q3[r].RT = Q3[r].BT;

```

```

100         Q3[r].name = Q2[i].name;
101         r = r + 1;
102         flag = 2;
103     }
104 }
105 {
106     if (flag == 2)
107         printf("\nProcess in third queue following FCFS ");
108     for (i = 0; i < r; i++)
109     {
110         if (i == 0)
111             Q3[i].CT = Q3[i].BT + time - tq1 - tq2;
112         else
113             Q3[i].CT = Q3[i - 1].CT + Q3[i].BT;
114     }
115     for (i = 0; i < r; i++)
116     {
117         Q3[i].TAT = Q3[i].CT;
118         Q3[i].WT = Q3[i].TAT - Q3[i].BT;
119         printf("\n%c\t%d\t%d\t%d\t", Q3[i].name, Q3[i].BT, Q3[i].WT, Q3[i].TAT);
120         sumW += Q3[i].WT;
121         sumT += Q3[i].TAT;
122         sumR += Q3[i].BT;
123     }
124     printf("\n\nAverage WT: %f\n", (float)sumW / (float)n);
125     printf("Average TAT: %f\n", (float)sumT / (float)n);
126     printf("Average RT: %f\n", (float)sumR / (float)n);
127 }
128
129 }
130

```

هذا الكود بلغة C يحاكي خوارزمية جدولة قائمة انتظار متعددة المستويات (MLFQ) بثلاث قوائم انتظار (Q1 و Q2 و Q3).

#### 1. هيكل البيانات:

- `struct process` : يعرف عملية بسمات مثل الاسم (`name`), وقت الوصول (`AT`), وقت التنفيذ (`BT`), وقت الانتظار (`WT`), زمن الاستجابة (`TAT`), الوقت المتبقى (`RT`), ووقت الانتهاء (`CT`).
- `process` : مصفوفات من نوع `Q1[10],Q2[10],Q3[10]` تمثل قوائم الانتظار الثلاث.
- `n` : يخزن عدد العمليات.

#### 2. دالة `sortByArrival()`:

هذه الدالة تقوم بترتيب العمليات في `Q1` بناءً على أوقات وصولها باستخدام خوارزمية ترتيب الفقاعات البسيطة. هذا يضمن معالجة العمليات بترتيب وصولها.

#### 3. دالة `main()`:

- المدخلات: يطلب من المستخدم إدخال عدد العمليات (`n`) ثم وقت الوصول ووقت التنفيذ لكل عملية. يتم تهيئة الوقت المتبقى (`RT`) إلى وقت التنفيذ.
- الترتيب: استدعاء دالة `sortByArrival()` لترتيب العمليات حسب وقت الوصول.
- التهيئة: تهيئة `time` إلى وقت وصول أول عملية.
- معالجة Q1 (دورانية مع 8):

التكرار خلال كل عملية في  $Q_1$  ▪

- إذا كان الوقت المتبقى ( $RT$ ) أقل من أو يساوي زمن الكوانتم ( $tq_1$ )، فإن العملية تكتمل في قائمة الانتظار هذه. يتم حساب وقت الانتظار وزمن الاستجابة والمقاييس الأخرى.

- إذا كان  $RT$  أكبر من  $tq_1$ ، يتم نقل العملية إلى  $Q_2$  بعد التنفيذ لمدة  $tq_1$  وحدة. يتم تحديث الوقت المتبقى، وتم إضافة العملية إلى  $Q_2$ . يتم تعين  $flag$  إلى 1 لإشارة إلى أن  $Q_2$  بها عمليات.

معالجة  $Q_2$  (دورانية مع  $16 = tq_2$ ) ▪

- إذا كانت  $flag$  تساوي 1 (يعني أن  $Q_2$  بها عمليات)، يتم تنفيذ هذا الجزء.

على غرار  $Q_1$ ، تتم معالجة العمليات في  $Q_2$  باستخدام خوارزمية دورانية بزمن كوانتم أكبر ( $tq_2$ ).

- إذا لم تكتمل عملية في  $Q_2$ ، يتم نقلها إلى  $Q_3$ . يتم تعين  $flag$  إلى 2.

معالجة  $Q_3$  (FCFS - من يأتي أولاً يُخدم أولاً) ▪

- إذا كانت  $flag$  تساوي 2 (يعني أن  $Q_3$  بها عمليات)، يتم تنفيذ هذا الجزء.

$Q_3$  تستخدم جدولة FCFS بسيطة. يتم حساب وقت انتهاء كل عملية بناءً على وقت انتهاء العملية السابقة. ثم يتم حساب وقت الانتظار وزمن الاستجابة.

الخرج: طباعة تفاصيل العملية (وقت التنفيذ، وقت الانتظار، زمن الاستجابة) لكل قائمة انتظار. أخيراً، يتم حساب وطباعة متوسط وقت الانتظار، متوسط زمن الاستجابة، ومتوسط زمن التنفيذ (الذي يعادل وقت التنفيذ في هذه الحالة لأنه محسوب على أنه  $(sumR/n)$ ).

تحسينات وتوضيحات رئيسية:

- إدارة قوائم الانتظار: يستخدم الكود ثلاث قوائم انتظار منفصلة ( $Q_1$  و  $Q_2$  و  $Q_3$ ) لإدارة العمليات بمستويات أولوية مختلفة.
- زمن الكوانتم: لكل قائمة انتظار زمن كوانتم خاص بها ( $8 = tq_1$ ،  $16 = tq_2$ ).
- خوارزميات الجدولة: يتم استخدام خوارزميات جدولة مختلفة لكل قائمة انتظار: دورانية لـ  $Q_1$  و  $Q_2$ ، و FCFS لـ  $Q_3$ .
- حساب المقاييس: يتم حساب وقت الانتظار، زمن الاستجابة، وזמן التنفيذ لكل عملية.
- الخرج: يتم تنظيم الخرج حسب قائمة الانتظار، مما يسهل فهم عملية الجدولة.