جامعة المنارة

كلية: الهندسة

قسم: المعلوماتية

اسم المقرر: نظم تشغيل 2

رقم الجلسة (2)

عنوان الجلسة

خوارزميات إدارة العمليات متعددة المستويات و جدولتها

الفصل الدراسي

الأول

العام الدراسي

2024_2023

جدول المحتويات

Contents

الغاية من الجلسة:

تعريف الطالب بخوارزميات توزيع العمليات Processes ضمن قائمة الانتظار إلى عدة مستويات حسب الأولوية المرتبطة بنوع العملية أو قوائم ديناميكية تتغير حسب زمن التنفيذ

خوارزميات قوائم الانتظار متعددة المستويات

Multilevel Queue Algorithm

تم إنشاء فئة خوارزميات الجدولة للحالات التي يتم فيها تصنيف العمليات إلى مجموعات مختلفة, على سبيل المثال، يتم إجراء تقسيم مشترك بين العمليات الأمامية (أو التفاعلية) Interactive Processes والعمليات الخلفية (أو الدفعية) Batch Processes. لهذين النوعين من العمليات متطلبات وقت استجابة مختلفة، وبالتالي قد يكون لهما احتياجات جدولة مختلفة. بالإضافة إلى ذلك، قد يكون للعمليات التفاعلية الأولوية على العمليات في الخلفية أو الدفعية.

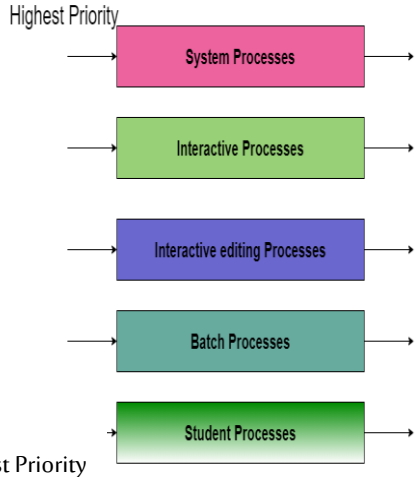
تقوم خوارزمية جدولة قائمة الانتظار متعددة المستويات بتقسيم قائمة الانتظار الجاهزة إلى عدة قوائم انتظار منفصلة. يتم تعيين العمليات بشكل دائم إلى قائمة انتظار واحدة، بناءً بشكل عام على بعض خصائص العملية، مثل حجم الذاكرة، أو أولوية العملية، أو نوع العملية. كل قائمة انتظار لها خوارزمية الجدولة الخاصة بها.

على سبيل المثال، يمكن استخدام قوائم انتظار منفصلة للعمليات التفاعلية والدفعية. قد تتم جدولة قائمة الانتظار الأمامية بواسطة خوارزمية Round Robin، بينما تتم جدولة قائمة انتظار الخلفية بواسطة خوارزمية FCFS.

بالإضافة إلى ذلك، يجب أن تكون هناك جدولة بين قوائم الانتظار، والتي يتم تنفيذها بشكل اعتيادي كجدولة وقائية ذات أولوية ثابتة. على سبيل المثال، قد تكون لقائمة الانتظار الأمامية أولوية مطلقة على قائمة انتظار في الخلفية.

يمكن تقسيم خوارزمية جدولة قائمة الانتظار متعددة المستويات إلى خمس قوائم انتظار:

- System Processes عمليات النظام
- Interactive Processes العمليات التفاعلية
- Interactive Editing Processes عمليات التحرير التفاعلية
- Batch Processes العمليات الدفعية
- Student Processes العمليات الطلابية



في هذه الحالة، إذا لم تكن هناك عمليات في قائمة الانتظار ذات الأولوية الأعلى فقط، فسيتم تشغيل العمليات الموجودة في قوائم الانتظار ذات الأولوية المنخفضة. على سبيل المثال: بمجرد أن تصبح العمليات الموجودة في قائمة انتظار النظام وقائمة الانتظار التفاعلية وقائمة انتظار التحرير التفاعلية فارغة، فسيتم تشغيل العمليات الموجودة في قائمة انتظار الدُفعات فقط.

وصف العمليات في الرسم البياني أعلاه هو كما يلي:

- عملية النظام

نظام التشغيل نفسه لديه عملية خاصة به للتشغيل ويطلق عليه اسم عملية النظام.

- عملية تفاعلية

العملية التفاعلية هي عملية تتطلب تفاعل بين الحاسب و المستخدم و تعطي نتيجة فورية .

- العمليات في الخلفية

المعالجة الدفعية هي في الأساس تقنية في نظام التشغيل تقوم بجمع البرامج والبيانات معًا في شكل دفعة قبل بدء المعالجة.

- عملية الطالب

تحظى عملية النظام دائمًا بالأولوية العليا بينما تحصل عمليات الطالب دائمًا على الأولوية الأدنى.

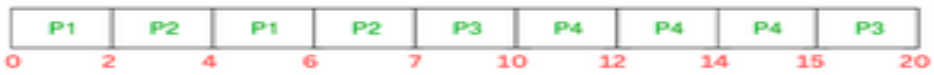
في نظام التشغيل، هناك العديد من العمليات، ومن أجل الحصول على النتيجة لا يمكننا وضع جميع العمليات في قائمة الانتظار؛ وبالتالي يتم حل هذه العملية عن طريق جدولة قائمة الانتظار متعددة المستويات.

مثال عملي :

خذ بعين الاعتبار الجدول أدناه الذي يضم أربع عمليات ضمن جدولة قائمة الانتظار متعددة المستويات. يشير رقم قائمة الانتظار إلى قائمة انتظار العملية

Process	Arrival Time	CPU Burst Time	Queue Number
P1	0	4	1
P2	0	3	1
P3	0	8	2
P4	10	5	1

أولوية قائمة الانتظار 1 أكبر من قائمة الانتظار 2. تستخدم قائمة الانتظار 1 Round Robin (الكم الزمني = 2) وتستخدم قائمة الانتظار 2 FCFS.



في البداية، يكون لكلا قائمتي الانتظار عملية، لذا يتم تشغيل العملية في قائمة الانتظار 1 (P1، P2) أولاً (بسبب الأولوية الأعلى) بطريقة الدورية وتكتمل بعد 7 وحدات

ثم تبدأ العملية في قائمة الانتظار 2 (P3) في التشغيل (حيث لا توجد عملية في قائمة الانتظار 1) ولكن أثناء تشغيلها، تأتي P4 في قائمة الانتظار 1 وتقاطع P3 وتبدأ في التشغيل لمدة 5 ثوانٍ و

بعد اكتماله، يأخذ P3 وحدة المعالجة المركزية ويكمل تنفيذها

تطبيق C ++ لجدولة قائمة الانتظار متعددة المستويات:

```
#<include>stdio.h
int main()
{
	int p[20],bt[20], su[20], wt[20],tat[20],i, k, n, temp;
	float wtavg, tatavg;
	printf("Enter the number of processes:”);
	scanf("%d",&n);
	for (i=0; i<n;i++)
	{
		p[i] = I;
		printf("Enter the Burst Time of Process%d:", i);
		scanf("%d”, &bt[i];
		printf("System/User Process (0/1)" ? );
		scanf("%d", &su[i]);
	}
	for(i=0;i<n;i++)
		for(k=i+1;k<n;k++)
			if(su[i] > su[k])
			{
			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;
			}
	wtavg = wt[0] = 0;
	tatavg = tat[0] = bt[0];
	for(i=1;i<n;i++)
	{
		wt[i] = wt[i-1] + bt[i-1];
		tat[i] = tat[i-1] + bt[i];
		wtavg = wtavg + wt[i];
		tatavg = tatavg + tat[i];
	}
	printf("\nPROCESS\t\t SYSTEM/USER PROCESS \tBURST TIME\tWAITING
TIME\tTURNAROUND TIME");
	for(i=0;i<n;i++)
		printf("\n%d \t\t %d \t\t %d \t\t %d \t\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”);
	return 0;
}
```

مزايا جدولة قائمة الانتظار متعددة المستويات

بمساعدة هذه الجدولة يمكننا تطبيق أنواع مختلفة من الجدولة لأنواع مختلفة من العمليات:

- بالنسبة لعمليات النظام: جدولة من يأتي أولاً يخدم أولاً (FCFS).
- بالنسبة للعمليات التفاعلية: جدولة أقصر مهمة أولاً (SJF).
- بالنسبة للعمليات المجمعة: جدولة RR(Round Robin).
- بالنسبة للعمليات الطلابية: جدولة الأولويات

عيوب جدولة قائمة الانتظار متعددة المستويات

العيب الرئيسي لجدولة قائمة الانتظار متعددة المستويات هو مشكلة بطئ التنفيذ أو عدم التنفيذ و هو ما يدعى بالتجويع للعمليات ذات المستوى الأدنى.

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

Multilevel Feedback Queue Scheduling

خوارزميات قوائم الانتظار المرنة متعددة المستويات

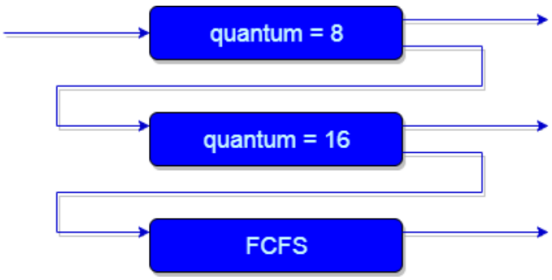
في خوارزمية جدولة قوائم الانتظار متعددة المستويات، يتم تعيين العمليات بشكل دائم إلى قائمة انتظار عند الدخول إلى النظام. تنتقل العمليات بين قوائم الانتظار. يتمتع هذا الإعداد بميزة انخفاض أعباء الجدولة، ولكن عيبه هو عدم المرونة.

ومع ذلك، فإن جدولة قائمة انتظار االمرنة متعددة المستويات تسمح للعملية بالتنقل بين قوائم الانتظار. تتمثل الفكرة في فصل العمليات المختلفة. إذا كانت العملية تستخدم الكثير من وقت وحدة المعالجة المركزية، فسيتم نقلها إلى قائمة انتظار ذات أولوية أقل. وبالمثل، قد يتم نقل العملية التي تنتظر لفترة طويلة جدًا في قائمة انتظار ذات أولوية أقل إلى قائمة انتظار ذات أولوية أعلى. هذا الشكل يمنع المجاعة.

بشكل عام، يتم تعريف جدولة انتظار الملاحظات متعددة المستويات من خلال المعلمات التالية:

- عدد قوائم الانتظار.
- خوارزمية الجدولة لكل قائمة انتظار.
- الطريقة المستخدمة لتحديد متى يتم ترقية العملية إلى قائمة انتظار ذات أولوية أعلى.
- الطريقة المستخدمة لتحديد متى يتم تخفيض عملية ما إلى قائمة انتظار ذات أولوية أقل.
- الطريقة المستخدمة لتحديد قائمة الانتظار التي ستدخلها العملية عندما تحتاج هذه العملية إلى الخدمة.

إن الجدولة المتعددة المستويات هي أكثر خوارزميات جدولة وحدة المعالجة المركزية عمومية. يمكن تهيئته ليتناسب مع نظام معين قيد التصميم. لسوء الحظ، يتطلب الأمر أيضًا بعض وسائل اختيار القيم لجميع المعلمات لتحديد أفضل جدولة. على الرغم من أن قائمة انتظار الملاحظات متعددة المستويات هي المخطط الأكثر عمومية، إلا أنها أيضًا الأكثر تعقيدًا.



يمكن رؤية مثال على قائمة انتظار التعليقات متعددة المستويات في الشكل أعلاه.

توضيح:

أولاً، لنفترض أن الصفين 1 و2 يتبعان جولة روبن مع الكم الزمني 8 و16 على التوالي، وأن الصف 3 يتبع FCFS.
أحد تطبيقات جدولة قائمة انتظار الملاحظات متعددة المستويات هو كما يلي:

إذا بدأت أي عملية في التنفيذ، فإنها تدخل أولاً في قائمة الانتظار 1.

في قائمة الانتظار 1، يتم تنفيذ العملية لـ 8 وحدات، وإذا اكتملت في هذه الوحدات الثمانية أو أنها توفر وحدة المعالجة المركزية (CPU) لتشغيل الإدخال / الإخراج في هذه الوحدات الثمانية، فإن أولوية هذه العملية لا تتغير، وإذا عادت لبعض الأسباب مرة أخرى في قائمة الانتظار الجاهزة ثم يبدأ التنفيذ مرة أخرى في قائمة الانتظار 1.

إذا لم تكتمل العملية الموجودة في قائمة الانتظار 1 في 8 وحدات، فسيتم تقليل أولويتها ويتم نقلها إلى قائمة الانتظار 2.

تنطبق النقطتان 2 و 3 أعلاه أيضًا على العمليات الموجودة في قائمة الانتظار 2 ولكن الكم الزمني هو 16 وحدة. بشكل عام، إذا لم تكتمل أي عملية في وقت معين، فسيتم نقلها إلى قائمة الانتظار ذات الأولوية الأقل.

بعد ذلك، في قائمة الانتظار الأخيرة، تتم جدولة جميع العمليات بطريقة FCFS.

من المهم ملاحظة أن العملية الموجودة في قائمة انتظار ذات أولوية أقل لا يمكن تنفيذها إلا عندما تكون قوائم الانتظار ذات الأولوية الأعلى فارغة.

يمكن مقاطعة أي عملية جارية في قائمة الانتظار ذات الأولوية المنخفضة عن طريق وصول عملية إلى قائمة الانتظار ذات الأولوية الأعلى.

أيضًا، قد يختلف التنفيذ أعلاه بالنسبة للمثال الذي ستتبع فيه قائمة الانتظار الأخيرة جدولة Round-robin.

في التنفيذ أعلاه هناك مشكلة وهي؛ أي عملية موجودة في قائمة الانتظار ذات الأولوية الأدنى يجب أن تعاني من المجاعة بسبب بعض العمليات القصيرة التي تستهلك كل وقت وحدة المعالجة المركزية.

والحل لهذه المشكلة هو :

هناك حل يتمثل في تعزيز أولوية كل العمليات بعد فترات زمنية منتظمة ثم وضع كافة العمليات في قائمة الانتظار ذات الأولوية الأعلى.

الحاجة إلى جدولة قائمة انتظار الملاحظات متعددة المستويات (MFQS)

فيما يلي بعض النقاط لفهم الحاجة إلى مثل هذه الجدولة المعقدة:

تعد هذه الجدولة أكثر مرونة من جدولة قائمة الانتظار متعددة المستويات.

تساعد هذه الخوارزمية في تقليل وقت الاستجابة.

من أجل تحسين وقت الاستجابة، هناك حاجة إلى خوارزمية SJF والتي تتطلب بشكل أساسي وقت تشغيل العمليات من أجل جدولتها. كما نعلم أن وقت تشغيل العمليات غير معروف مسبقًا. كما أن هذه الجدولة تعمل بشكل أساسي على تشغيل عملية لكمية زمنية وبعد ذلك يمكنها تغيير أولوية العملية إذا كانت العملية طويلة. وبالتالي، تتعلم خوارزمية الجدولة هذه بشكل أساسي من السلوك السابق للعمليات ومن ثم يمكنها التنبؤ بالسلوك المستقبلي للعمليات. بهذه الطريقة، تحاول MFQS تشغيل عملية أقصر أولاً مما يؤدي في المقابل إلى تحسين وقت الاستجابة.

مزايا MFQS

هذه خوارزمية جدولة مرنة

تسمح خوارزمية الجدولة هذه للعمليات المختلفة بالتنقل بين قوائم الانتظار المختلفة.

في هذه الخوارزمية، قد يتم نقل العملية التي تنتظر لفترة طويلة جدًا في قائمة انتظار ذات أولوية أقل إلى قائمة انتظار ذات أولوية أعلى مما يساعد في منع المجاعة.

عيوب MFQS

هذه الخوارزمية معقدة للغاية.

نظرًا لأن العمليات تتحرك حول قوائم انتظار مختلفة مما يؤدي إلى إنتاج المزيد من النفقات العامة لوحدة المعالجة المركزية.

من أجل اختيار أفضل جدولة، تتطلب هذه الخوارزمية بعض الوسائل الأخرى لتحديد القيم

برنامج C ++ لمحاكاة جدولة انتظار ردود الفعل متعددة المستويات

يتكون برنامج جدولة انتظار الملاحظات متعدد المستويات Q من 3 قوائم انتظار خطية، أي Q1 وQ2 وQ3.

- Q1 هو RR مع الزمن 5 (RR5)

- Q2 هو RR مع الزمن 8 (RR8)

- Q3 يتبع من يأتي أولاً يخدم أولاً (FCFS)

- لا يمكن تنفيذ العملية في قائمة الانتظار السفلية في حالة وجود أي وظائف في جميع المهام الأعلى

طوابير. على سبيل المثال، يحتوي Q1 على 5 عمليات، وQ2 لديه عملية واحدة، وQ3 لديه عملية واحدة يجب أولاً تنفيذ العملية في Q1 (وإكمالها)، ثم يتم تنفيذ العملية في Q2. وأخيرا، سيحصل Q3 على مورد وحدة المعالجة المركزية.

- تدخل عملية جديدة في قائمة الانتظار Q1 والتي يتم تقديمها إلى RR5.

• عندما تحصل على وحدة المعالجة المركزية (CPU)، تتلقى العملية 5 مللي ثانية.

• إذا لم تنتهي خلال 5 مللي ثانية، يتم نقل العملية إلى قائمة الانتظار Q2.

• في عملية Q2 يتم تقديم RR8 مرة أخرى ويستقبل 8 ميلي ثانية إضافية.

• إذا لم يكتمل بعد، فسيتم استباقه ونقله إلى قائمة الانتظار Q3.

• في Q3 يتم تنفيذ العملية على أساس من يأتي أولاً يخدم أولاً.

• إذا لم يكتمل بعد، فستتم معالجته في Q2 حتى اكتماله.

الخرج:

يتم عرض الوقت المتبقي للعمليات في كل مستوى من مستويات قائمة الانتظار وإجمالي وقت الانتظار وإجمالي وقت الاستجابة.

```
struct process
{
   char name;
   int AT,BT,WT,TAT,RT,CT;
}Q1[10],Q2[10],Q3[10];/*Three queues*/

int n;
void sortByArrival()
{
struct process temp;
int i,j;
for(i=0;i<n;i++)
```

```
   {
      for(j=i+1;j<n;j++)
         {
            if(Q1[i].AT>Q1[j].AT)
               {
                    temp=Q1[i];
                    Q1[i]=Q1[j];
                    Q1[j]=temp;
               }
         }
   }
}

int main()
{
    int i,j,k=0,r=0,time=0,tq1=5,tq2=8,flag=0;
    char c;
    printf("Enter no of processes:");
    scanf("%d",&n);
    for (i=0, c='A'; i<n;i++,c++)
    {
       Q1[i].name=c;
       printf("\nEnter the arrival time and burst time of process %c: ",Q1[i].name);
       scanf("%d%d", &Q1[i].AT,&Q1[i].BT);
       Q1[i]. RT=Q1[i].BT;/*save burst time in remaining time for each process*/

   }
sortByArrival();
time=Q1[0].AT;
printf("Process in first queue following RR with qt=5");
printf("\nProcess\t\tRT\t\tWT\t\tTAT\t\t");
for(i=0;i<n;i++)
{

 if(Q1[i].RT<=tq1)
 {

      time+=Q1[i].RT;/*from arrival time of first process to completion of this process*/
      Q1[i].RT=0;
      Q1[i].WT=time-Q1[i].AT-Q1[i].BT;/*amount of time process has been waiting in
the first queue*/
      Q1[i].TAT=time-Q1[i].AT;/*amount of time to execute the process*/
      printf("\n%c\t\t%d\t\t%d\t\t%d",Q1[i].name,Q1[i].BT,Q1[i].WT,Q1[i].TAT);

 }
 else/*process moves to queue 2 with qt=8*/
 {
    Q2[k].WT=time;
    time+=tq1;
```

```
      Q1[i].RT-=tq1;
      Q2[k].BT=Q1[i].RT;
      Q2[k].RT=Q2[k].BT;
      Q2[k].name=Q1[i].name;
      k=k+1;
      flag=1;
   }
}
if(flag==1)
 {printf("\nProcess in second queue following RR with qt=8");
  printf("\nProcess\t\tRT\t\tWT\t\tTAT\t\t");
}for(i=0;i<k;i++)
   {
    if(Q2[i].RT<=tq2)
     {
       time+=Q2[i].RT;/*from arrival time of first process +BT of this process*/
       Q2[i].RT=0;
       Q2[i].WT=time-tq1-Q2[i].BT;/*amount of time process has been waiting in the
ready queue*/
       Q2[i].TAT=time-Q2[i].AT;/*amount of time to execute the process*/
       printf("\n%c\t\t%d\t\t%d\t\t%d",Q2[i].name,Q2[i].BT,Q2[i].WT,Q2[i].TAT);

    }
    else/*process moves to queue 3 with FCFS*/
    {
      Q3[r].AT=time;
      time+=tq2;
      Q2[i].RT-=tq2;
      Q3[r].BT=Q2[i].RT;
      Q3[r].RT=Q3[r].BT;
      Q3[r].name=Q2[i].name;
      r=r+1;
      flag=2;
    }
  }

{if(flag==2)
printf("\nProcess in third queue following FCFS ");
}
for(i=0;i<r;i++)
{
    if(i==0)
          Q3[i].CT=Q3[i].BT+time-tq1-tq2;
       else
          Q3[i].CT=Q3[i-1].CT+Q3[i].BT;

}

for(i=0;i<r;i++)
```

```
    {
        Q3[i].TAT=Q3[i].CT;
        Q3[i].WT=Q3[i].TAT-Q3[i].BT;
        printf("\n%c\t\t%d\t\t%d\t\t%d\t\t",Q3[i].name,Q3[i].BT,Q3[i].WT,Q3[i].TAT);

    }

}
```