جامعة المنارة

كلية: .....الهندسة.....

قسم: ..... الهندسة المعلوماتية.....

اسم المقرر: ..... نظم تشغيل 2.....

رقم الجلسة (...6...)

عنوان الجلسة

....جدولة القرص الصلب.....

م.عمار مصطفى م.فاطمة جندي

الفصل الدراسي العام الدراسي

2025/ 2024

جدول المحتويات

Contents

الغاية من الجلسة: التعرف على خوارزميات جدولة القرص الصلب

خوارزميات جدولة القرص الصلب تُستخدم في أنظمة التشغيل لتحديد الترتيب الذي تُخدم به طلبات الوصول إلى القرص الصلب (HDD) أو وحدات التخزين الأخرى، بهدف تقليل زمن البحث (Seek Time) وزيادة الكفاءة. رأس القرص (Disk Head) يتحرك عبر المسارات (Tracks) للوصول إلى البيانات، واختيار الخوارزمية المناسبة يؤثر على الأداء.

1- أنواع خوارزميات جدولة القرص الصلب

1-1- الخدمة حسب الترتيب الأولي (First-Come, First-Served) FCFS :

- ✧ تُخدم الطلبات حسب ترتيب وصولها.
- ✧ بسيطة لكن غير فعالة إذا كانت الطلبات موزعة بشكل عشوائي، لأنها قد تؤدي إلى حركة كبيرة لرأس القرص.

1-2- الأقرب أولاً (Shortest Seek Time First) SSTF :

- ✧ يتم اختيار الطلب الأقرب لرأس القرص الحالي (أقل زمن بحث).
- ✧ فعالة لتقليل حركة الرأس، لكن قد تؤدي إلى التجويع (Starvation)لطلبات بعيدة.

1-3- المسح (Elevator Algorithm) SCAN :

- ✧ رأس القرص يتحرك في اتجاه واحد (مثلاً من الداخل إلى الخارج) ويخدم كل الطلبات في طريقه، ثم يعكس الاتجاه.
- ✧ يضمن عدم التجويع، لكن قد يتأخر الطلب إذا كان في الاتجاه المعاكس.

1-4- المسح الدائري (Circular SCAN) C-SCAN :

- ✧ نسخة محسنة منSCAN ، يتحرك الرأس في اتجاه واحد فقط، وعندما يصل إلى النهاية يعود مباشرة إلى البداية دون خدمة الطلبات أثناء العودة.
- ✧ يوفر توزيعًا أكثر عدالة للطلبات.

5-1- خوارزمية LOOK وC-LOOK

- ✧ مشابهة لـ SCAN وC-SCAN، لكن الرأس لا يذهب إلى نهاية القرص إذا لم تكن هناك طلبات، بل يعكس الاتجاه عند آخر طلب.
- ✧ أكثر كفاءة من SCAN و C-SCANلأنها تقلل الحركة غير الضرورية.

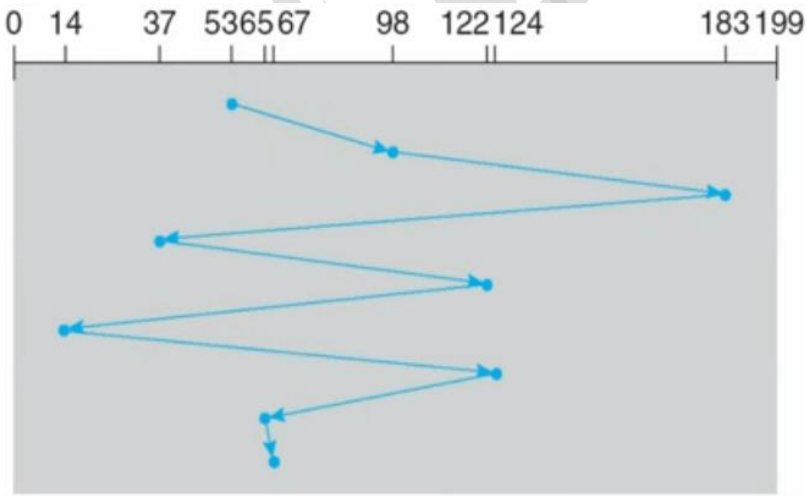
مثال عملي

لنفترض أن لدينا قرصًا به 200 مسار (0 إلى 199)، ورأس القرص في المسار 53، والطلبات التالية في قائمة الانتظار:

طلبات المسارات 98 ، 183، 37، 122، 14، 124، 65، 67

الهدف :حساب إجمالي حركة الرأس (عدد المسارات) باستخدام خوارزميات مختلفة.

FCFS (First-Come, First-Served):



الترتيب: 53 ⟶ 98 ⟶ 183 ⟶ 37 ⟶ 122 ⟶ 14 ⟶ 124 ⟶ 65 ⟶ 67

الحركة :

53 ⟶ 98: |98 - 53| = 45

98 ⟶ 183: |183 - 98| = 85

183 ⟶ 37: |183 - 37| = 146

37 ⟶ 122: |122 - 37| = 85

122 ⟶ 14: |122 - 14| = 108

14 ⟶ 124: |124 - 14| = 110

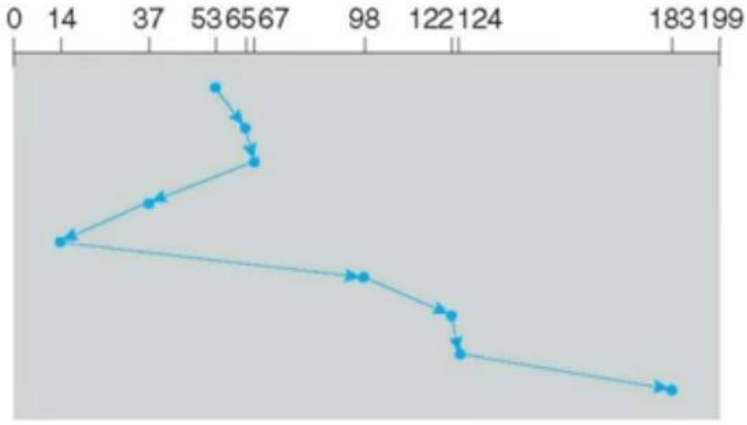
124 ⟶ 65: |124 - 65| = 59

65 ⟶ 67: |67 - 65| = 2

الإجمالي مسارًا: 45 + 85 + 146 + 85 + 108 + 110 + 59 + 2 = 640

SSTF (Shortest Seek Time First):

نبدأ من 53 ونختار الأقرب دائمًا :



65 ⟶ 53(الأقرب لـ 53): |53 - 65| = 12

65 ⟶ 67: |67 - 65| = 2

67 ⟶ 98: |98 - 67| = 31

98 ⟶ 122: |122 - 98| = 24

122 ⟶ 124: |124 - 122| = 2

124 ⟶ 183: |183 - 124| = 59

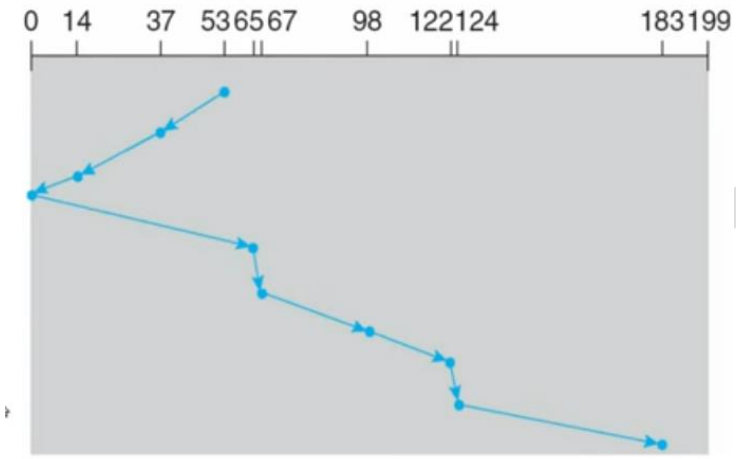
183 ⟶ 37: |183 - 37| = 146

37 ⟶ 14: |37 - 14| = 23

الإجمالي مسارًا: 12 + 2 + 31 + 24 + 2 + 59 + 146 + 23 = 299

SCAN(باتجاه الأعلى، من 53 إلى 199 ثم العودة):

الطلبات مرتبة: 14، 37، 65، 67، 98، 122، 124، 183



من 53 إلى 199: 53 ⟶ 65 ⟶ 67 ⟶ 98 ⟶ 122 ⟶ 124 ⟶ 183

53 ⟶ 65: 12

65 ⟶ 67: 2

67 ⟶ 98: 31

98 ⟶ 122: 24

122 ⟶ 124: 2

124 ⟶ 183: 59

يصل إلى 199 (لا طلبات أخرى): |199 - 183| = 16

يعكس الاتجاه إلى 0، يخدم 37 و14 :

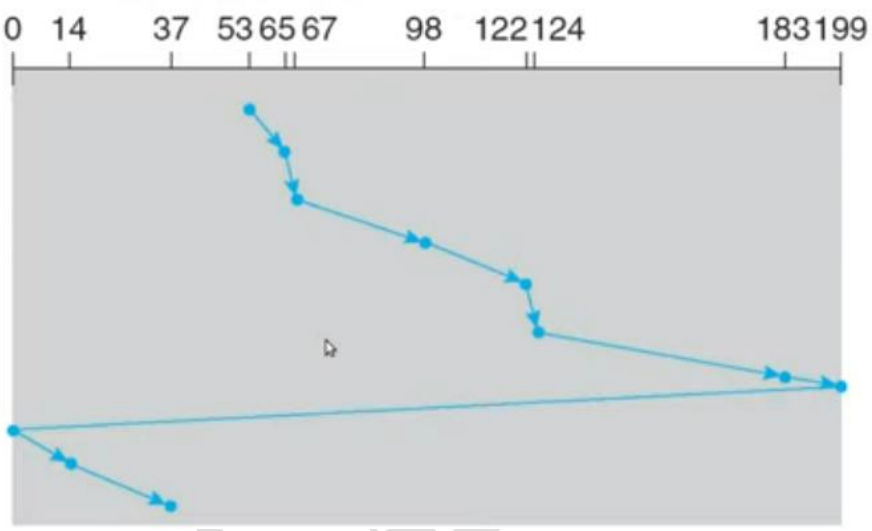
199 ⟶ 37: 162

37 ⟶ 14: 23

الإجمالي مسارًا: 12 + 2 + 31 + 24 + 2 + 59 + 16 + 162 + 23 = 331

C-SCAN(باتجاه الأعلى، من 53 إلى 199 ثم العودة إلى 0):

من 53 إلى 199: 53 ⟶ 65 ⟶ 67 ⟶ 98 ⟶ 122 ⟶ 124 ⟶ 183



53 ⟶ 65: 12

65 ⟶ 67: 2

67 ⟶ 98: 31

98 ⟶ 122: 24

122 ⟶ 124: 2

124 ⟶ 183: 59

يصل إلى 199، ثم يعود إلى 0: |199 - 183| + |199 - 0| = 16 + 199 = 215

من 0 إلى 53: يخدم 14 و37

0 ⟶ 14: 14

14 ⟶ 37: 23

الإجمالي 382 = 23 + 14 + 199 + 16 + 59 + 2 + 24 + 31 + 2 + 12 :مسارًا

LOOK(مثل SCAN لكن يتوقف عند آخر طلب)

من 53 إلى 183: 53 ⟶ 65 ⟶ 67 ⟶ 98 ⟶ 122 ⟶ 124 ⟶ 183

53 ⟶ 65: 12

65 ⟶ 67: 2

67 ⟶ 98: 31

98 ⟶ 122: 24

122 ⟶ 124: 2

124 ⟶ 183: 59

يعكس الاتجاه عند 183 إلى 14 :

183 ⟶ 37: 146

37 ⟶ 14: 23

الإجمالي 299 = 23 + 146 + 59 + 2 + 24 + 31 + 2 + 12 :مسارًا

مقارنة بين الخوارزميات

FCFS: 640(غير فعال بسبب الحركة العشوائية).

SSTF: 299(فعال لكنه قد يسبب التجويع).

SCAN: 331(عدالة أكثر لكنه يشمل حركة إضافية إلى 199).

C-SCAN: 382(أكثر عدالة لكنه أقل كفاءة منSCAN).

LOOK: 299(مثل SCAN لكن بدون حركة غير ضرورية إلى 199).

2- كيف نختار خوارزميات جدولة الأقراص:

- **إذا كنت تبحث عن البساطة** :اختر **FCFS**، لكن كن مستعدًا لأداء ضعيف إذا كانت الطلبات عشوائية.
- **إذا كان الهدف تقليل زمن البحث** :اختر **SSTF**أو **LOOK**، لكن راقب التجويع إذا كانت الطلبات موزعة على نطاق واسع.
- **إذا كنت تهتم بالعدالة** :اختر **SCAN**أو .**C-SCAN**إذا كنت تريد عدالة مع تقليل الحركة الزائدة، اختر **C-LOOK**.
- **للأنظمة ذات التحميل العالي C-SCAN** :أو **C-LOOK**هما الأفضل لضمان توزيع عادل وزمن استجابة معقول.

مثال عملي لاتخاذ القرار

لنفترض أنك تدير خادوم قاعدة بيانات (Database Server) مع طلبات موزعة: 98، 183، 37، 122، 14، 124، 65، 67، ورأس القرص في المسار 53.

- **تحليل الوضع :**
  - الطلبات موزعة بشكل عشوائي (من 14 إلى 183).
  - الخادوم يخدم عدة مستخدمين، لذا العدالة مهمة.
  - زمن الاستجابة مهم أيضًا لضمان تجربة مستخدم جيدة.
- **الاختيار :**
  - **SSTF** (299 مسارًا) فعالة لتقليل الحركة، لكن قد تتسبب في تجويع الطلب 183.
  - **LOOK** (299 مسارًا أيضًا) جيدة لأنها تحقق نفس الكفاءة مع ضمان خدمة الطلبات في كلا الاتجاهين.
  - **C-SCAN** (382 مسارًا) تضمن عدالة أكبر لأنها تعامل جميع الطلبات بشكل دائري، وهو مناسب للخادوم.
- **القرار C-LOOK** :هو الخيار الأمثل هنا، لأنه يجمع بين العدالة(مثلC-SCAN) وتقليل الحركة غير الضرورية (مثلLOOK).

تطبيق برمجي:

برنامج C++ لتنفيذ خوارزمية **SSTF (Shortest Seek Time First)** (رأس القرص في المسار 53 والطلبات: 98، 183، 37، 122، 14، 124، 65، 67).

```
#include <iostream>

#include <climits>

using namespace std;

int main() {

  int head = 53; // موقع رأس القرص الابتدائي

  int requests[] = {98, 183, 37, 122, 14, 124, 65, 67}; // طلبات المسارات
```

```
const int n = 8; // عدد الطلبات
int sequence[8]; // مصفوفة لتخزين تسلسل الطلبات
int totalSeekTime = 0; // إجمالي زمن البحث
bool visited[8] = {false}; // مصفوفة لتتبع الطلبات المخدمة
int current = head; // موقع الرأس الحال
// تكرار لخدمة جميع الطلبات
for (int i = 0; i < n; i++) {
  int minSeek = INT_MAX;// تهيئة الحد الأدنى لزمن البحث بأكبر قيمة ممكنة
  int nextTrackIndex = -1; // لهدف من هذه التهيئة هو التأكد من أننا لم نحدد طلبًا بعد، وسنقوم بتحديث هذه القيمة داخل الحلقة عندما نجد الطلب الأقرب.
  // البحث عن الطلب الأقرب
  for (int j = 0; j < n; j++) {
    if (!visited[j]) { // تحقق مما إذا كان الطلب لم يُخدم بعد
      int seek = abs(requests[j] - current); // حساب مسافة البحث بين موقع الرأس الحالي والطلب
      if (seek < minSeek) {
        minSeek = seek;
        nextTrackIndex = j;
      }}}
  // تخزين الطلب التالي في التسلسل
  sequence[i] = requests[nextTrackIndex]; // تخزين الطلب الحالي في تسلسل الخدمة
  visited[nextTrackIndex] = true; // وضع علامة على الطلب كمخدم
  totalSeekTime += minSeek; // إضافة زمن البحث
  current = requests[nextTrackIndex]; // تحديث موقع الرأس
```

```
    }
    // عرض النتائج
    cout << "Sequence of track access: ";
    for (int i = 0; i < n; i++) {
        cout << sequence[i];
        if (i < n - 1) cout << " -> "; }
    cout << endl;
    cout << "Total Seek Time: " << totalSeekTime << " tracks" << endl;
    cout << "Average Seek Time: " << (float)totalSeekTime / n << " tracks" << endl;
    return 0;
}
```