

The Variance Principle:

1. 

The lifetime of a task is the time between the start of the task and the completion of the task.

2. 

The time quantum is the amount of time allocated to a task before it is preempted.

The lifetime of a task is the time between the start of the task and the completion of the task.

The time quantum is the amount of time allocated to a task before it is preempted.

The lifetime of a task is the time between the start of the task and the completion of the task.

The time quantum is the amount of time allocated to a task before it is preempted.

The lifetime of a task is the time between the start of the task and the completion of the task.

The time quantum is the amount of time allocated to a task before it is preempted.

The lifetime of a task is the time between the start of the task and the completion of the task.

The time quantum is the amount of time allocated to a task before it is preempted.

The lifetime of a task is the time between the start of the task and the completion of the task.

The time quantum is the amount of time allocated to a task before it is preempted.
The document contains a diagram and text in Hebrew. The text discusses the concept of mutual interference in the context of field theory and mathematical logic. The diagram illustrates the relationship between sets and their intersection and union, which is a key concept in understanding mutual interference.

The text is broken into paragraphs, each discussing different aspects of the topic. The paragraphs are written in a formal mathematical style, using symbols and notation commonly used in theoretical computer science and logic.

The document concludes with a summary of the main points, restated in a more accessible language. The summary highlights the importance of understanding mutual interference and its implications for the field of computer science and logic.
Process P;
repeat
  while (turn != i) {* wait *};
  CS
  turn := j;
  RS
forever
Shared data:
choosing: array[0..n-1] of boolean; // initialized to false.
number: array[0..n-1] of integer; // initialized to 0

Process P0:
repeat
  choosing[i] := true;
  number[i] := max(number[0]..number[n-1]) + 1;
  choosing[i] := false;
  for j := 0 to n-1 do {
    while (choosing[j]) {
      while (number[j] != 0 and (number[j], j) < (number[i], i)) {};
    }
  }
CS
number[i] := 0;
RS
forever

bool testset(int &i)
{
  if (i == 0) {
    i = 1;
    return true;
  } else return false;
}

Process P:
repeat
  repeat {} until testset(b);  
  CS
  b := 0;
  RS
forever

\[ P(S) \begin{cases} 
  \text{if } S < 0 & \text{then } \text{put process in wait} \\
  \text{else } S = S - 1 
\end{cases} \]

\[ V(S) \begin{cases} 
  S = S + 1 
\end{cases} \]
A busy-wait \( P(S) \) or \( V(S) \) becomes boomer and continues to check the condition \( S = 0 \) as long as the process is running.

```
Initialization:
S.count := 1; in := 0;
N.count := 0; out := 0;
E.count := k;

append(v):
| b[in] := v; in := (in + 1) mod k; |
| repeat |
| produce v; |
| wait(E); |
| wait(S); |
| signal(v); |
| signal(N); |
| forever |

Consumer:
repeat
| wait(N); |
| wait(S); |

take():
w := b[OuT]; out := (out + 1) mod k; return w;
```

The Dining Philosophers Problem

In a dining room, 5 philosophers sit around a table and have 5 forks. Each philosopher needs 2 forks to eat. To solve this problem, we can use a critic or a wait function to check whether both forks are available before using them.

```
Process P:
repeat
think;
wait(stick[i]);
wait(stick[i+1 mod 5]);
eat;
signal(stick[i+1 mod 5]);
signal(stick[i]);
```

stick: array[0..4] of semaphores
Initialization:
stick[i].count := 1 for i:= 0..4

The Dining Philosophers Problem

Starvation vs. deadlock

If the philosophers are always waiting for forks, they may starve.

```
repeat
think;
wait(stick[i]);
wait(stick[i+1 mod 5]);
eat;
signal(stick[i+1 mod 5]);
signal(stick[i]);
```

deadlock

If the forks are always available, they may deadlock.

```
repeat
think;
wait(stick[i]);
wait(stick[i+1 mod 5]);
eat;
signal(stick[i+1 mod 5]);
signal(stick[i]);
```

The Dining Philosophers Problem

Starvation vs. deadlock

If the philosophers are always waiting for forks, they may starve.

```
repeat
think;
wait(stick[i]);
wait(stick[i+1 mod 5]);
eat;
signal(stick[i+1 mod 5]);
signal(stick[i]);
```

deadlock

If the forks are always available, they may deadlock.

```
repeat
think;
wait(stick[i]);
wait(stick[i+1 mod 5]);
eat;
signal(stick[i+1 mod 5]);
signal(stick[i]);
```

The Dining Philosophers Problem

Starvation vs. deadlock

If the philosophers are always waiting for forks, they may starve.

```
repeat
think;
wait(stick[i]);
wait(stick[i+1 mod 5]);
eat;
signal(stick[i+1 mod 5]);
signal(stick[i]);
```

deadlock

If the forks are always available, they may deadlock.

```
repeat
think;
wait(stick[i]);
wait(stick[i+1 mod 5]);
eat;
signal(stick[i+1 mod 5]);
signal(stick[i]);
```
stick: array[0..4] of semaphores
Initialization:
    stick[i].count := 1 for i := 0..4
    T.count := 4

Process P:
    repeat
        think;
        wait(T);
        wait(stick[i]);
        wait(stick[i+1 mod 5]);
        eat;
        signal(stick[i+1 mod 5]);
        signal(stick[i]);
        signal(T);
    forever

Предикат "Deadlock"

Предикат "Deadlock"

ШПП(с1, ..., сn)

for i := 1 to n do
    Si := Si + 1;
    S := 0
    for j := 1 to n do
        if Si >= j
            then Si := 0;
        else S := S + 1;
    end for
    if S = n
        then deadlock := true;
    else deadlock := false;
end for

ШПП(с1, ..., сn)

if (с2 >= 1 and ..., and сn >= 1)
    then for i := 1 to n do
        Si := Si + 1;
    end for
else
    begin
        if Si < 1
            then deadlock := true;
        else S := 1:
    end if

диаграмма

относительно этой ситуации

של המקרה 5 הפיתולוסים יצרו顺丰 שילוט התoultry

call (test and set)

call (atomite)
**رمز של מחשבים**

埒שליפיקו פיתוח של בודק עם עיר ממולא ל"כלי" سيكون כפישון. קר 2 פיסוק יעוד: 1. תessment ו-2. Test and Set.


4 MULTI programming - 1. לא עובד  ו-2..Util.

**רמות עץ יעוד**

ǒ - not immediate ,oil - or immediate logic_submit (משלים: Machine Check) – test set and set.


**ề של דק**

odesk רשת לא מתאימה שטח ממולא בודק סדרתי: ENQ/DEQ.

ניקי ונCalibriים שמשלים שטח סדרתיים ב-2 רמות: 1. אסכולרמר (קר לא שמשלים מבוקש). 2. ניתן להגשים בודק.SHARED.

cי צמצום על ממולאים? מבוקש מעין uc 2 שמות (קטינים: "למשל: הסכם שמות מקдоб" uc, להכרה: "למשל: הסכם שמות מקдоб" uc, module) בודק ואתים, Ayrıca מחליפים את

**_decrypt**

" based on bucket sorting" באוניברסיטת שמים איזנים ל-k בודק אתים מחליפים את ל-k בודק אתים.

טוק ול->k בודק אתים.

**פיסוק בודק ומלא**

ENQ - סדרת


cng qname, rname, \( e_s \), rname_length, \{ system \}, ret = \{ test \}

systems use none

(systios: "פש הסpatches." ו- qname, rname_length, \( e_s \), name)

name_length, \( e_s \) – name מפריך. דוד מתאמה \( 255 \) קבץ יש פרוסי של \( a \) שמשלים בודק בודק איזון חסיד: בודק אתים איזונים בודק אתים בודק אתים (systems)
shared-free change = change

test

test

\[\text{DEQ}\]

multi-recordMessage

 tease.HQ

MessagePass

send, receive

send(dest, msg), receive(source, msg): msg = \{'name': 'major QCB', 'name': 'minor QCB'}

[1, 'send', 'send']

[2, 'receive']

[3, 'blocked']

FQ

Locks


test

\[\text{lock}\]
לerialized - type

<table>
<thead>
<tr>
<th>מספר ה- TYPE</th>
<th>קבעו משירות בפורמה הבאה:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>dispatcher</td>
</tr>
<tr>
<td>2</td>
<td>asm (address space manager)</td>
</tr>
<tr>
<td>3</td>
<td>malloc</td>
</tr>
<tr>
<td>4</td>
<td>ioscates</td>
</tr>
<tr>
<td>5</td>
<td>isosuch</td>
</tr>
<tr>
<td>6</td>
<td>iosch</td>
</tr>
<tr>
<td>7</td>
<td>srm</td>
</tr>
<tr>
<td>8</td>
<td>local</td>
</tr>
</tbody>
</table>

- **lock**
- **cond**
- **cond disable**
- **cond disabled**

**ס bruk**

- **Consumable**
- **Reentrant**

**מעון הקימה**

ל师兄 הקימה ה-ן-ן-ן-

**זמן בחרון**

- **Deadlock**
- ** Serialization**

**ס bruk**

- **deadlock**
- **deadlock-enabled**
- **deadlock-disabled**

**צuko**

1.
2.

http://www.cs.biu.ac.il/~pinraz/
Swapping

If the job succeeds, the
new job replaces the old
job.

always

If the job fails, the
reserved time for the
new job is cancelled.

Swapping

Job

send
recv
recv

if

if

Deadlock

Guard and the
conditions for
 Creating a
deadlock

Guard:

- Mutual exclusion
- Wait condition
- No preemption
- Circular wait

Conditions for creating a deadlock:

- [P_1, ..., P_n] - set of processes
- {R_1, ..., R_m} - set of resources

Deadlock:

- One or more processes are blocked
- All processes are waiting for resources
- No process can proceed

To prevent deadlocks:

- Use resource allocation graphs
- Identify cycles in the graph
- Implement deadlock prevention mechanisms

References:

1. 
2. 
3. 
4. 
5.
deadlock

: prevention – mutual exclusion.

deadlock: prevention - mutual exclusion. 

1. deadlocks are caused by the deadlock in the system. 

2. deadlocks can be prevented by using mutual exclusion. 

3. A set of processes in the system is said to be deadlock-free if no process in the set can ever result in a deadlock.

4. A system is deadlock-free if it is impossible for any process to be in the set described in 3.

: detection and recovery

1. When a deadlock occurs, it is detected by checking the state of the system.

2. Once a deadlock is detected, it is resolved by choosing one of the processes involved in the deadlock and freeing its resources.

: prevention

1. deadlocks can be prevented by using mutual exclusion.

2. A set of processes in the system is said to be deadlock-free if no process in the set can ever result in a deadlock.

3. A system is deadlock-free if it is impossible for any process to be in the set described in 2.

: deadlock

1. A system is said to be deadlock-free if it is impossible for any process to be in the set described in 2.

2. Deadlocks can be prevented by using mutual exclusion.

3. A set of processes in the system is said to be deadlock-free if no process in the set can ever result in a deadlock.

4. A system is deadlock-free if it is impossible for any process to be in the set described in 3.

: prevention

1. deadlocks can be prevented by using mutual exclusion.

2. A set of processes in the system is said to be deadlock-free if no process in the set can ever result in a deadlock.

3. A system is deadlock-free if it is impossible for any process to be in the set described in 2.

: detection and recovery

1. When a deadlock occurs, it is detected by checking the state of the system.

2. Once a deadlock is detected, it is resolved by choosing one of the processes involved in the deadlock and freeing its resources.

: prevention

1. deadlocks can be prevented by using mutual exclusion.

2. A set of processes in the system is said to be deadlock-free if no process in the set can ever result in a deadlock.

3. A system is deadlock-free if it is impossible for any process to be in the set described in 2.

: deadlock

1. A system is said to be deadlock-free if it is impossible for any process to be in the set described in 2.

2. Deadlocks can be prevented by using mutual exclusion.

3. A set of processes in the system is said to be deadlock-free if no process in the set can ever result in a deadlock.

4. A system is deadlock-free if it is impossible for any process to be in the set described in 3.

: detection and recovery

1. When a deadlock occurs, it is detected by checking the state of the system.

2. Once a deadlock is detected, it is resolved by choosing one of the processes involved in the deadlock and freeing its resources.

: prevention

1. deadlocks can be prevented by using mutual exclusion.

2. A set of processes in the system is said to be deadlock-free if no process in the set can ever result in a deadlock.

3. A system is deadlock-free if it is impossible for any process to be in the set described in 2.

: deadlock

1. A system is said to be deadlock-free if it is impossible for any process to be in the set described in 2.

2. Deadlocks can be prevented by using mutual exclusion.

3. A set of processes in the system is said to be deadlock-free if no process in the set can ever result in a deadlock.

4. A system is deadlock-free if it is impossible for any process to be in the set described in 3.

: detection and recovery

1. When a deadlock occurs, it is detected by checking the state of the system.

2. Once a deadlock is detected, it is resolved by choosing one of the processes involved in the deadlock and freeing its resources.

: prevention

1. deadlocks can be prevented by using mutual exclusion.

2. A set of processes in the system is said to be deadlock-free if no process in the set can ever result in a deadlock.

3. A system is deadlock-free if it is impossible for any process to be in the set described in 2.

: deadlock

1. A system is said to be deadlock-free if it is impossible for any process to be in the set described in 2.

2. Deadlocks can be prevented by using mutual exclusion.

3. A set of processes in the system is said to be deadlock-free if no process in the set can ever result in a deadlock.

4. A system is deadlock-free if it is impossible for any process to be in the set described in 3.

: detection and recovery

1. When a deadlock occurs, it is detected by checking the state of the system.

2. Once a deadlock is detected, it is resolved by choosing one of the processes involved in the deadlock and freeing its resources.
avoidance

A. 3.1 avoidance

In the previous section, we introduced the concept of "avoidance" in the context of animal behavior. Avoidance behavior is characterized by an organism's attempt to avoid a harmful stimulus or situation. This can be achieved through a variety of strategies, such as fleeing, freezing, or seeking shelter.

In this section, we will delve deeper into the biological and psychological mechanisms underlying avoidance behavior. We will explore the role of the amygdala and other brain regions in processing threat signals, as well as the role of cognitive processes in shaping avoidance behavior.

B. 3.2 Avoidance and Anxiety

The concept of avoidance can be extended to human behavior as well. In clinical psychology, avoidance is a common symptom of anxiety disorders, such as social anxiety disorder and post-traumatic stress disorder. Avoidance behaviors in these cases serve to reduce anxiety by avoiding situations or stimuli that are perceived as threatening.

Understanding the mechanisms underlying avoidance behavior is crucial for developing effective treatment strategies. In the next section, we will discuss some of the latest research findings in this area, including the role of neuroimaging and cognitive-behavioral therapy in identifying and treating avoidance behavior.
4. **Switching Methods:**
   - Circuit Switching
   - Message Switching
   - Packet Switching
   - Cell Switching

5. **Multicast Bus**
   - Multi Access Bus
   - Full Access Bus

6. **Data Transfer:**
   - Switching Methods and Their Use
   - Multicast Message Switching

7. **Collision Detection:**
   - Ethernet: Collision Detection
   - Token Passing

8. **Ethernet-B:**
   - Frame Format
   - Ethernet-C:
השבט הארוך.

ARPANET

קביעת: 1. שיפוט ATP L2, LAN 1

APRINET

 wooded העץ הפרפרים של LAN

שבט מחשבים

file server

שמורת כ健康产业

AIX

NFS שים בעברית

AIX

AIX

ARPANET

אריזה: האammers המнатים מעל הקביצים במערכת, ב- 206, זו המוקדerta במערכת המחשבים או ATM

השבט הארוך.

ARPANET

קביעת: 1. שיפוט ATP L2, LAN 1

APRINET

 wooded העץ הפרפרים של LAN

שבט מחשבים

file server

שמורת כ健康产业

AIX

NFS שים בעברית

AIX

AIX

ARPANET

אריזה: האammers המнатים מעל הקביצים במערכת, ב- 206, זו המוקדerta במערכת המחשבים או ATM

השבט הארוך.
The document appears to be in Hebrew. It discusses DCE (Distributed Computing Environment), its features, and its components. It mentions topics such as recovery, data sharing, LAN, and WAN. The text also references RFC 2 and RFC 3. The document seems to be a technical or academic paper on distributed computing. No figures or tables are present in the document.