CS571 Operating Systems

CLASS 5 : 24 SEP 2003
16:30 – 19:10

Except where otherwise noted, these slides are © copyright Charles Snow, 2003. Permission to copy these slides is granted so long as each slide contains the copyright notice “© C. M. Snow 2003” appears on each page and the copies are made freely available, i.e., not sold. Students are advised that they should not use University consumables (ink, toner, paper) to produce printed versions of these slides. Rights for all other use of this material are reserved by the author; permission for any other use must be obtained in writing from the author.

Point of Sale Terminal

• standard POS terminal resembles:
  - keyboard
  - bar code scanner
  - internal logic
  - printer
  - display

Terminals

• a regular computer terminal:
  - keyboard
  - input encoding logic
  - UART
  - serial out
  - display driver logic
  - screen

Terminals: OS-eye view

• user types key
• bits arrive serially at UART in computer’s serial port
• when all bits received, generate interrupt
• kernel picks up byte of data from serial port and sends it...

UNIX Processes and ttys

• each process that is started at a terminal has associated with it a controlling tty
  - i.e., UNIX treats tty as a single device that is read/write capable
• each process has 3 files open for it:
  - stdin: where input is expected
  - stdout: where output is expected to go
  - stderr: where error output is expected to go
UNIX Processes and ttys

- each process that is started at a terminal has associated with it a controlling tty
- each process has 3 files open for it
- normally, a process begun from a controlling tty has:
  - stdin = keyboard
  - stdout, stderr = display

UNIX ttys

- ttys normally operate in full duplex mode:
  - consequence: received characters must be echoed back to display in order for user to see them
  - ⇒ tty driver program must perform echo operation
- other things this program does?

UNIX ttys

- line buffering: programs reading stdin only get complete line-at-a-time,
  - e.g., fread(mybuffer, 1, 256, stdin) returns only when a complete line (256 chars) is read
  - or EOF, or error

Normal tty settings

- terminal characteristics: (use "stty –a" command)
  - speed 38400 baud, rows = 67, columns = 108,

  - special chars:
    - intr = ^c  quit = ^A  erase = ^k  kill = ^u
    - eof = ^d  eol = ^?  eol2 = ^]  switch = ^c
    - start = ^q  stop = ^z  susp = ^t  dosusp = ^y
    - rpre = ^r  flush = ^o  werase = ^w  lnext = ^u

  - modes:
    - parenb -parodd cs8 -cstopb -hupcl -cread -
    - ignbrk -brkint -igncar -parenb -inpck -istrip -
    -ixon -ixany -lsxof -mizs -sizs -icanon -
    - echo -echonl -echoe -echock -tostop -echost -
    - echiport -echok -echoke -flusho -oppost - بلغة -
    - onlcr -ocrnl -onocr -onlret -ofill -ofdel -

© Charles M Snow 2
Normal tty settings

• all aspects of tty operation are controllable
  • via appropriate system calls

UNIX device control

• for ttys use termios functions (man termios)
  • e.g.,
    • int tcgetattr(int fd, struct termios *termios_p);
    • int tcsetattr(int fd, int optional_actions, struct termios *termios_p);
  • use for, e.g., control of echoing, character interpretation, ...

UNIX devices

• all devices are represented as files
  • may be opened or closed
  • may be read and/or written
  • may be "controlled"
    • e.g., seek command on a disk drive

  • int fd = open(filename,flags,mode);
    full path to file to be opened, e.g., dev/hdai
    read, write, r/w, etc.
    setting access modes

Deadlock

• 2 processes each waiting for the other to release a resource being held so processes can advance
  • generalizes to ≥ 2 processes, ≥ 2 resources

• problem occurs when processes follow a request→use→release model of resource utilization
  • i.e., exclusive use

UNIX device control

• basic system call:
  • int ioctl(int fd, req, ...);

• valid requests depends on type of device (see, e.g., man ioctl_list)

• many categories of device have ioctl() wrappers
  • easier to work with

Deadlock

• resources may be hardware
  • e.g., memory
  • assume 200 Kb of memory are available:

<table>
<thead>
<tr>
<th></th>
<th>P1</th>
<th></th>
<th>P2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>...</td>
<td>request 80 Kb</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>request 60 Kb</td>
<td>...</td>
<td>request 80 Kb</td>
</tr>
</tbody>
</table>
Deadlock

- resources may be software
  - e.g., resources created/destroyed as needed, like signals, messages, data buffers

<table>
<thead>
<tr>
<th>P1</th>
<th>P2</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>receive(P2,M)</td>
<td>receive(P1,R)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>send(P2,N)</td>
<td>send(P1,R)</td>
</tr>
</tbody>
</table>

Conditions for deadlock

1. mutual exclusion
   - only one process may use resource at a time

2. hold and wait
   - a process may hold allocated resources while waiting to receive others requested

3. no pre-emption
   - no resource may be forcibly taken from a process holding it

Depicting Deadlock

- resource allocation graphs
  - nodes:
    - type P for processes
    - type R for resources
  - edges:
    - request P → R
    - assignment R → P

Conditions for deadlock

1. mutual exclusion
   - only one process may use resource at a time

2. hold and wait
   - a process may hold allocated resources while waiting to receive others requested

3. no pre-emption
   - no resource may be forcibly taken from a process holding it

4. circular wait
   - a closed chain of processes such that each holds ≥ 1 needed by next process in chain
Depicting Deadlock: RAGs

- a resource class may have $\geq 1$ instance of that resource type
- show as dots within rectangle for resource node

$R_1 \cdot R_3 \cdot R_4 \cdots R_2 \cdot \cdots P_1 P_2 P_3$ Figure 8.1 in SG&G

Depicting Deadlock: RAGs

- no cycle here: no deadlock

$R_1 \cdot R_3 \cdot R_4 \cdots R_2 \cdot \cdots P_1 P_2 P_3$ Figure 8.1 in SG&G

Depicting Deadlock: RAGs

- absence of a cycle in a RAG $\Rightarrow$ no deadlock
- presence of a cycle $\Rightarrow$ is deadlock?

$R_1 \cdot \cdots R_2 \cdot \cdots P_1 P_2 P_3 P_4$ Figure 8.2 in SG&G

Depicting Deadlock: RAGs

- now cycle here: now deadlock

$R_1 \cdot \cdots R_2 \cdot \cdots P_1 P_2 P_3$ Figure 8.3 in SG&G

Depicting Deadlock: RAGs

- absence of a cycle in a RAG $\Rightarrow$ no deadlock
- presence of a cycle $\Rightarrow$ is deadlock?
  - if each R has only 1 instance, then yes
  - otherwise: may mean deadlock, may not

$R_1 \cdot \cdots R_2 \cdot \cdots P_1 P_2 P_3$ Figure 8.3 in SG&G
Deadlock: What Can You Do?

- no really satisfying answers, but:
  1. prevention
  2. detection
  3. avoidance

Preventing Deadlock

- hold & wait: prevent by
  1. require a process to request all of its resources before it begins running
  2. block the process until all its requests can be granted simultaneously
  3. or
     a process requesting resources is not allowed to be holding any
     if it has some and wants more, it must first release all held resources, then request

Preventing Deadlock

- prevention: prevent any of the four conditions from occurring
  1. indirect: prevent any of conditions 1, 2, or 3
  2. direct: prevent condition 4

Preventing Deadlock

- disallow mutual exclusion
  1. not really feasible
  2. the nature of some resources precludes sharing them: must be used one-at-a-time

⇒ reject as solution

Preventing Deadlock

- no pre-emption: allow pre-emption
  1. a process requesting resources may be denied and have resources taken away from it
  2. it must re-acquire these later to continue
  3. or
     a process already holding resources is denied further requests because another process is already waiting for [other] resources
     we pre-emptively take resources away from waiting proc and give them to new requestor

⇒ works, but not really useful for many resources
Preventing Deadlock

- circular wait:
  - assign an ordering to resource types
  - require requests to be made in order
  - does this work?
    - let \( i < j \), so \( R_i \) precedes \( R_j \)
    - let \( P_A \) have \( R_i \) and need \( R_j \)
    - hence have deadlock
    - \( i < j \) and \( i > j \)
    - works at preventing deadlock

Detecting Deadlock

- what kind of algorithm?
  - algorithm able to find cycles in graphs could find deadlock
  - start with RAC:
    - delete \( R \) nodes
    - collapse assignment edges accordingly
    - result is wait-for-graph (WFG)
    - if there is a cycle in WFG, there is deadlock

Detecting Deadlock

- idea:
  - let any process request and hold any resource any time
  - i.e., let deadlock occur if it’s going to
  - run some algorithm to detect such deadlock
  - what kind of algorithm?

RAGs to WFGs

Based on slide 05.30
Deadlock Detected: now what?

- so you detect deadlock, now what?
- several strategies possible:

Deadlock Detected, now what?

1. abort all deadlocked processes

Deadlock Detected, now what?

2. roll-back each deadlocked process to some previous checkpoint, then re-run them

Deadlock Detected, now what?

3. abort deadlocked processes one by one until deadlock goes away
   - use some “minimal-cost” metric to guide selection of processes

Deadlock Detected, now what?

1. abort all deadlocked processes

Deadlock Detected, now what?

2. roll-back each deadlocked process to some previous checkpoint, then re-run them

Deadlock Detected, now what?

3. abort deadlocked processes one by one until deadlock goes away
   - use some “minimal-cost” metric to guide selection of processes
   - least CPU used so far
   - lowest priority
   - highest estimated time remaining

Deadlock Detected, now what?

1. abort all deadlocked processes

Deadlock Detected, now what?

2. roll-back each deadlocked process to some previous checkpoint, then re-run them

Deadlock Detected, now what?

3. abort deadlocked processes one by one until deadlock goes away
   - use some “minimal-cost” metric to guide selection of processes
   - after each proc is killed, need to re-run detection algorithm to test if done

© Charles M Snow
Deadlock Detected, now what?

1. abort all deadlocked processes
2. roll-back each deadlocked process to some previous checkpoint, then re-run them
3. abort deadlocked processes one by one until deadlock goes away
4. pre-empt resources until deadlock goes away
   • roll process back to point before it acquired resource

Deadlock Avoidance

- prevention
  • very conservative approach
  • assures no deadlock
  • may have high efficiency penalty
- detection
  • deadlock can occur
  • runs expensive detection algorithm
  • deals with any detected deadlock
- avoidance – strategy in the middle
  • focuses on condition (4) and makes wise choices

Deadlock Avoidance

Let \( R \) be the resource vector:

\[
R_1 \quad R_2 \quad R_3 \quad \cdots \quad R_m
\]

Total amount of each resource in system

Let \( V \) be the available vector:

\[
V_1 \quad V_2 \quad V_3 \quad \cdots \quad V_m
\]

Total amount of each resource not allocated to a process

Deadlock Avoidance

Let \( C \) be the claim matrix:

\[
C_{11} \quad \cdots \quad C_{1m} \\
C_{21} \quad \cdots \quad C_{2m} \\
C_{31} \quad \cdots \quad C_{3m} \\
\vdots \quad \cdots \quad \vdots \\
C_{n1} \quad \cdots \quad C_{nm}
\]

Maximum requirements of each process for each resource:

\( C_{ij} \) is requirement of process \( i \) for resource \( j \)

Deadlock Avoidance

Let \( A \) be the allocation matrix:

\[
A_{11} \quad \cdots \quad A_{1m} \\
A_{21} \quad \cdots \quad A_{2m} \\
A_{31} \quad \cdots \quad A_{3m} \\
\vdots \quad \cdots \quad \vdots \\
A_{n1} \quad \cdots \quad A_{nm}
\]

Currently allocated resources

\( A_{ij} \) is quantity of resource \( j \) now allocated to process \( i \)
Deadlock Avoidance

- with these vectors and matrices, we have:
  - all resources are either allocated or available:
    \[ R_i = V_i + \sum_{k=1}^{n} A_{ki} \] for all \( i \)
  - no process can claim more than the total amount of resources now available: \( C_{ki} \leq R_i \) for all \( k, i \)
  - no process is allocated more resources of any type than it originally claimed it needed: \( A_{ki} \leq C_{ki} \) for all \( k, i \)

Banker's Algorithm

- a state of this system is the current allocation of resources as represented by \( R, V, C \) and \( A \)
- a state is:
  - safe: there is \( \geq 1 \) order in which all procs can be run to completion without deadlock
  - unsafe: there is no order in which all procs ...
- idea: when a proc asks for a resource, see if granting that resource would result in a safe or unsafe state
  - block requesting proc if result would be unsafe

Deadlock Avoidance

- OK to start a new process iff: \( R \geq \sum_{k=1}^{n} C_{ki} + C_{(n+1)i} \)
  - i.e., begin only if maximum claim of all current \( n \) processes plus those of the new process can be met
  - process initiation denial
  - always assumes worst case (everyone wants everything at once), so may needlessly delay a process

Banker's Algorithm

- Djikstra, 1965
- a resource denial algorithm
- consider a system with:
  - fixed number of processes
  - fixed number of resources

Djikstra, 1965
- a resource denial algorithm
- consider a system with:
  - fixed number of processes
  - fixed number of resources

- can \( P_1 \) finish?
  - no: cannot grant all its resource requests
### Banker's Algorithm

#### Claim matrix

<table>
<thead>
<tr>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>P2</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>P3</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>P4</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>P2</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>P3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>P4</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

#### Allocation matrix

<table>
<thead>
<tr>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>P2</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>P3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>P4</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

**Claim matrix**

<table>
<thead>
<tr>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>P2</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>P3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>P4</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Allocation matrix**

<table>
<thead>
<tr>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>P2</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>P3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>P4</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

**Claim matrix**

<table>
<thead>
<tr>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>P2</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>P3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>P4</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Allocation matrix**

<table>
<thead>
<tr>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>P2</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>P3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>P4</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

#### Claim matrix

<table>
<thead>
<tr>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>P2</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>P3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>P4</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Allocation matrix**

<table>
<thead>
<tr>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>P2</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>P3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>P4</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

#### Claim matrix

<table>
<thead>
<tr>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>P2</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>P3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>P4</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Allocation matrix**

<table>
<thead>
<tr>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>P2</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>P3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>P4</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

#### Claim matrix

<table>
<thead>
<tr>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>P2</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>P3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>P4</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Allocation matrix**

<table>
<thead>
<tr>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>P2</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>P3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>P4</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

#### Claim matrix

<table>
<thead>
<tr>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>P2</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>P3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>P4</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Allocation matrix**

<table>
<thead>
<tr>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>P2</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>P3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>P4</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

#### Claim matrix

<table>
<thead>
<tr>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>P2</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>P3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>P4</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Allocation matrix**

<table>
<thead>
<tr>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>P2</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>P3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>P4</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

**Claim matrix**

<table>
<thead>
<tr>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>P2</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>P3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>P4</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Allocation matrix**

<table>
<thead>
<tr>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>P2</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>P3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>P4</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

### Questions

- **can P2 finish?**
  - **yes**: grant 1 more R3 then P2 runs to completion releasing 6 × R1, 1 × R2 and 3 × R3

- **with P2 finished,**
  - P1 now can be granted its requests and finish releasing 3 × R1, 2 × R2 and 2 × R3

- **with P2, P1, and P3 finished,**
  - P4 now can be granted its requests and finish releasing 4 × R1, 2 × R2 and 2 × R3 every process now finished, so, this is a safe state

- **would granting P1’s request for 1 × R1 and 1 × R3 result in a safe state?**

- **if we do the grant we get this: is this a safe state?**
Banker’s Algorithm

<table>
<thead>
<tr>
<th></th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>P2</td>
<td>6</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>P3</td>
<td>3</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>P4</td>
<td>4</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Claim matrix

<table>
<thead>
<tr>
<th></th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>P2</td>
<td>5</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>P3</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>P4</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

Allocation matrix

<table>
<thead>
<tr>
<th></th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>9</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>R2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

if we do the grant we get this: is this a safe state? no: no process can complete
e.g., all need more R1’s but there are none and none can be freed

Avoidance with Banker’s Algorithm

- deadlock avoidance = staying in a safe state
- Banker’s algorithm does not predict deadlock with certainty
  - an unsafe state is not guaranteed to produce deadlock
  - an unsafe state has the potential to result in deadlock: that is what Banker’s avoids

- costs of using Banker’s:
  1. maximum resource requirements for each process must be known and stated in advance
  2. processes must be independent: order in which they run is unimportant

Avoidance with Banker’s Algorithm

- costs of using Banker’s:
  1. maximum resource requirements for each process must be known and stated in advance
  2. processes must be independent: order in which they run is unimportant
  3. must be a fixed number of both resources and processes

Observation on Deadlock Handling

- Tanenbaum notes:
  "...the schemes described earlier under the name 'prevention' are overly restrictive, and the algorithm described here as 'avoidance' requires information that is usually not available. If you can think of a general-purpose algorithm that does the job in practice as well as in theory, write it up and send it to your local computer science journal."
- there is one other approach...

© Charles M Snow 12
Ostrich Algorithm

- stick your head in the sand and pretend there’s no such thing as deadlock
- practical:
  - no cost for memory or processing overhead
  - deadlock is a sufficiently rare occurrence
- UNIX uses this approach
  - no detection, avoidance or prevention
  - so deadlocks can happen
- suitable for settings where deadlocks are rare and can be tolerated even when they occur

Practical:
- no cost for memory or processing overhead
- deadlock is a sufficiently rare occurrence
- UNIX uses this approach
  - no detection, avoidance or prevention
  - so deadlocks can happen
- suitable for settings where deadlocks are rare and can be tolerated even when they occur

Programs

- how do you run a program?
- program exists as disk file
  - called a load module
  - contains executable code
- how get file contents into memory?
  - need an OS service that loads load modules into memory: the loader
  - loader loads load modules into memory

Where do load modules come from?

- evolution of a load module (e.g., for C programs):
  - source code
  - header files
  - compiler
  - assembler code
  - assembler
  - object code
  - library object files
  - load module

Building a Load Module: CPP

```bash
/usr/lib/gcc-lib/i586-mandrake-linux-gnu/2.96/cpp0
-lang-c
-o .gcc_
-o .gcc_XHOR__96
-o .gcc_PATCHLEVEL__0
-o .elf_
-o .oH
-o .oL
-o .oL
-o .oL
-o .oL
-o .oL
-o .oL
-o .oL
-o .oL
-a system(posix)
-o_NO_ENDIAN
-o system(186)
-o machine(186)
```

End of search list.
Building a Load Module: CC1

```
/usr/lib/gcc-lib/i586-mandrake-linux-gnu/2.96/cc1
/tmp/ccmEBvF0.i
-quiet
-dumpbase
helloWorld.c
-version
-o /tmp/cc514J0.s
```

GNU C version 2.96 20000731 (Mandrake Linux 8.2 2.96-0.76mdk)
(Mandrake Linux 8.2 2.96-0.76mdk) compiled by GNU C version 2.96
20000731 (Mandrake Linux 8.2 2.96-0.76mdk).

Evolution of a load module

• start with simple:
  ```
  main() {
    printf("Hello world\n");
  }
  ```
  length: 44 bytes

Building a Load Module: AS

```
as
-\v
-\y
-o /tmp/cc514J0.o
```

GNU assembler version 2.11.92.0.12 (i586-mandrake-linux-gnu)
using BFD version 2.11.92.0.12 20011121

Evolution of a load module

• compiler produces (partial listing):
  ```
  .string "Hello world\n"
  .text
  .globl main
  .type main, @function
  main:
  pushl $ebp
  movl %esp, %ebp
  subl $8, %esp
  subl $12, %esp
  pushl $.LC0
  call printf
  addl $16, %esp
  movl %ebp, %esp
  popl %ebp
  ret
  ```
  on linux platform
  file size: 399 bytes

Building a Load Module: link

```
/usr/lib/gcc-lib/i586-mandrake-linux-gnu/2.96/collect2
-m elf_i386
-static
-helloWorld
/usr/lib/gcc-lib/i586-mandrake-linux-gnu/2.96/../../../crt1.o
/usr/lib/gcc-lib/i586-mandrake-linux-gnu/2.96/../../../crti.o
/usr/lib/gcc-lib/i586-mandrake-linux-gnu/2.96/crtbegin.o
-s/usr/lib/gcc-lib/i586-mandrake-linux-gnu/2.96/../../../crtend.o
```

Evolution of a load module

• assembler generates object file
  • on linux, file size is 936 bytes
  • on Sun Sparc (Solaris), file size is 784 byte

  in the .o file, we find (as reported by nm command):
  ```
  00000000 T main
  ```
  printf is an Undefined symbol in this object file
  main is a defined symbol in this object file Text segment
Evolution of a load module

- linker’s job: to resolve such undefined references by linking library object code with our newly created object code
  - or making such linking possible in future...
- printf() is in C run-time library (on Linux, e.g., libstdc++-3.1.2-2.10.0.a)
- linker generates load module:
  - on Linux, file size is 1,727,238 bytes
  - on Sun Sparc (Solaris), file size is 374,264 bytes

Different Kinds of Load Module

- absolute
  - problem: associated a real memory address to variables, targets at compile time

Different Kinds of Load Module

- load modules are built to suit the way the loader loads them into memory
- absolute:
  - real address used to refer to
    - locations holding variable values, e.g., CLR 05A21BE4
    - jump targets, e.g., JMP 4AF0328C
  - program must be loaded into real memory at exactly the location it was built for

Different Kinds of Load Module

- relocatable:
  - solution: delay that association until load time
  - generate code assuming module starts at location 0
  - compute all memory references relative to this start location
  - loader “fixes” relative-to-0 references when loading
    - requires extra information provided by compiler and/or assembler
    - produces a load module “bound” to the location where it is loaded

Different Kinds of Load Module

- absolute
  - so if load module swapped out, then back in to different address: badness
  - problem: associated a real memory address to variables, targets at load time
Different Kinds of Load Module

- dynamic run-time:
  - solution: delay that association until run-time
  - load program into real memory keeping relative addresses as they are
  - absolute addresses are calculated as needed at run time
    - using special hardware (to be talked about later)

Linkage

before:

```
00000000 _main: clr r0
00001044 jsr _printf
00002AE0 ret
00000000 _printf: mov r0, -(sp)
```

after:

```
00000000 _main: clr r0
00001044 jsr _printf
00002AE4 _printf: mov r0, -(sp)
```

Static Linking

- after assembly (with `.o` file):
  - relative object code for each invoked function is appended to code generated from assembler
  - linkage done on concatenated set of object components
  - produces one self-contained blob of code
    - may be big

Dynamic Linking

- link at load time
  - load module contains unresolved external references
  - locate and link object code for needed functions as load module is loaded into memory
    - adjust as needed while loading, relative to overall load module start
Dynamic Linking

- link at run time
  - call to function not already linked-in causes that function to be linked at that moment
  - e.g., printf() would be linked only when it is actually invoked

Dynamic Linking (run time): Pros and Cons

- benefits from new or updated libraries without re-link
- sharing of libraries in memory
- smaller executable image: only has functions actually used
- not fully self-contained
- less portable: needs libraries to be available
- may not work with updated libraries
- performance degradation on first “missed” invocation

Static Linking: Pros and Cons

- fully self-contained
- immune to library changes
- more portable
- very large executable image
- change to program ⇒ full re-link of everything

Dynamic Linking (load time): Pros and Cons

- benefits from new or updated libraries without re-link
- sharing of libraries in memory
- not fully self-contained
- less portable: needs libraries to be available
- may not work with updated libraries