CS341: Operating System

**Deadlock & Main Memory**
Lect 26: 08\textsuperscript{th} Oct 2014
Dr. A. Sahu  
Dept of Comp. Sc. & Engg.  
Indian Institute of Technology Guwahati

**Outline**
- Deadlock Conditions  
  - Mutex, Hold & Wait, No-Preemption and Circular wait
- Deadlock Prevention and Avoidance  
  - RAG, Cycle
- Deadlock Detection and Recovery  
  - WFG and RAG
- Main Memory and Memory Management

**Deadlock Detection**
- Allow system to enter deadlock state
- Detection algorithm
- Recovery scheme

**Single Instance of Each Resource Type**
- Maintain wait-for graph  
  - Nodes are processes  
  - \( P_i \rightarrow P_j \) if \( P_i \) is waiting for \( P_j \)
- Periodically searches for a cycle in WFG  
  - If there is a cycle, there exists a deadlock
- Detect a cycle in a graph  
  - \( O(n^2) \) operations, where \( n = \)number of vertices

**Resource-Allocation Graph and Wait-for Graph**

**Several Instances of a Resource Type**
- Similar to Safety Algorithm of Deadlock Avoidance
  - Available[1:M]: Currently Available  
  - Allocation[N:M]: currently allocated  
  - Request[N:M]: current request
**Deadlock Detection Graphical Approach**

- Resource Allocation Graph
- Reduction (Erase)
  - If a resource has only arrow away from it
    - No request pending to resource
  - If a process has only arrow pointing towards it
    - All the request granted
  - If a process has arrows pointing away from it but each such request arrow there is an available dot in resource: Erase all the process arrow

**Example**

If a process has only arrow pointing towards it
All the request granted

If a process has arrows pointing away from it but each such request arrow there is an available dot in resource: Erase all the process arrow
Example

If a process has arrows pointing away from it but each such request arrow there is an available dot in resource : Erase all the process arrow

No Deadlock

Another Example

Another Example

Another Example

Another Example

Another Example

Another Example

If a resource has only arrow away from it No request pending to resource

No-further reduction: There must be cycle
Several Instances of a Resource Type

- Similar to Safety Algorithm of Deadlock Avoidance
- Available[1:M]: Currently Available
- Allocation[N:M]: currently allocated
- Request[N:M]: current request

Detection Algorithm

Let Work[1:M] and Finish[1:M]
Work[1:M] = Available[1:M]; Found=false;
for (i = 0 ; i < n; i++) { if (Allocation, [1:M] ≠ 0)
Finish[i] = false; else Finish[i] = true; }
while(found==true)(
Find an index i such that both:
if (Found==false) break;
Work = Work + Allocation, Finish[i] = true
if (Finish[i] == false, for some i, 1 ≤ i ≤ n) {
the system is in deadlock state.
//Moreover, if Finish[i] == false, then P, is deadlocked

Example of Detection Algorithm

- Five processes P₀ through P₅, three resource types A (7 instances), B (2 instances), and C (6 instances)
- Snapshot at time T₀:

<table>
<thead>
<tr>
<th></th>
<th>Allocation</th>
<th>Request</th>
<th>Available</th>
</tr>
</thead>
<tbody>
<tr>
<td>P₀</td>
<td>0 1 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
</tr>
<tr>
<td>P₁</td>
<td>2 0 0</td>
<td>2 0 2</td>
<td></td>
</tr>
<tr>
<td>P₂</td>
<td>3 0 3</td>
<td>0 0 0</td>
<td></td>
</tr>
<tr>
<td>P₃</td>
<td>2 1 1</td>
<td>1 0 0</td>
<td></td>
</tr>
<tr>
<td>P₄</td>
<td>0 0 2</td>
<td>0 0 2</td>
<td></td>
</tr>
</tbody>
</table>

- Sequence <P₀, P₁, P₂, P₃, P₄> will result in Finish[i] = true for all i

Example of Detection Algorithm

- Five processes P₀ through P₅, three resource types A (7 instances), B (2 instances), and C (6 instances)
- Snapshot at time T₁:

<table>
<thead>
<tr>
<th></th>
<th>Allocation</th>
<th>Request</th>
<th>Available</th>
</tr>
</thead>
<tbody>
<tr>
<td>P₀</td>
<td>0 1 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
</tr>
<tr>
<td>P₁</td>
<td>2 0 0</td>
<td>2 0 2</td>
<td></td>
</tr>
<tr>
<td>P₂</td>
<td>3 0 3</td>
<td>0 0 0</td>
<td></td>
</tr>
<tr>
<td>P₃</td>
<td>2 1 1</td>
<td>1 0 0</td>
<td></td>
</tr>
<tr>
<td>P₄</td>
<td>0 0 2</td>
<td>0 0 2</td>
<td></td>
</tr>
</tbody>
</table>

- Sequence <P₀, P₁, P₂, P₃, P₄> will result in Finish[i] = true for all i

Example of Detection Algorithm

- Five processes P₀ through P₅, three resource types A (7 instances), B (2 instances), and C (6 instances)
- Snapshot at time T₂:

<table>
<thead>
<tr>
<th></th>
<th>Allocation</th>
<th>Request</th>
<th>Available</th>
</tr>
</thead>
<tbody>
<tr>
<td>P₀</td>
<td>0 1 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
</tr>
<tr>
<td>P₁</td>
<td>2 0 0</td>
<td>2 0 2</td>
<td></td>
</tr>
<tr>
<td>P₂</td>
<td>3 0 3</td>
<td>0 0 0</td>
<td></td>
</tr>
<tr>
<td>P₃</td>
<td>2 1 1</td>
<td>1 0 0</td>
<td></td>
</tr>
<tr>
<td>P₄</td>
<td>0 0 2</td>
<td>0 0 2</td>
<td></td>
</tr>
</tbody>
</table>

- Sequence <P₀, P₁, P₂, P₃, P₄> will result in Finish[i] = true for all i

Example of Detection Algorithm

- Five processes P₀ through P₅, three resource types A (7 instances), B (2 instances), and C (6 instances)
- Snapshot at time T₃:

<table>
<thead>
<tr>
<th></th>
<th>Allocation</th>
<th>Request</th>
<th>Available</th>
</tr>
</thead>
<tbody>
<tr>
<td>P₀</td>
<td>0 1 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
</tr>
<tr>
<td>P₁</td>
<td>2 0 0</td>
<td>2 0 2</td>
<td></td>
</tr>
<tr>
<td>P₂</td>
<td>3 0 3</td>
<td>0 0 0</td>
<td></td>
</tr>
<tr>
<td>P₃</td>
<td>2 1 1</td>
<td>1 0 0</td>
<td></td>
</tr>
<tr>
<td>P₄</td>
<td>0 0 2</td>
<td>0 0 2</td>
<td></td>
</tr>
</tbody>
</table>

- Sequence <P₀, P₁, P₂, P₃, P₄> will result in Finish[i] = true for all i

Example of Detection Algorithm

- Five processes P₀ through P₅, three resource types A (7 instances), B (2 instances), and C (6 instances)
- Snapshot at time T₄:

<table>
<thead>
<tr>
<th></th>
<th>Allocation</th>
<th>Request</th>
<th>Available</th>
</tr>
</thead>
<tbody>
<tr>
<td>P₀</td>
<td>0 1 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
</tr>
<tr>
<td>P₁</td>
<td>2 0 0</td>
<td>2 0 2</td>
<td></td>
</tr>
<tr>
<td>P₂</td>
<td>3 0 3</td>
<td>0 0 0</td>
<td></td>
</tr>
<tr>
<td>P₃</td>
<td>2 1 1</td>
<td>1 0 0</td>
<td></td>
</tr>
<tr>
<td>P₄</td>
<td>0 0 2</td>
<td>0 0 2</td>
<td></td>
</tr>
</tbody>
</table>

- Sequence <P₀, P₁, P₂, P₃, P₄> will result in Finish[i] = true for all i
Example of Detection Algorithm

- Five processes $P_0$ through $P_4$: three resource types
  - $A$ (7 instances), $B$ (2 instances), and $C$ (6 instances)

- Available
  - Snapshots

<table>
<thead>
<tr>
<th>Allocation</th>
<th>Request</th>
<th>Available</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$</td>
<td>$B$</td>
<td>$C$</td>
</tr>
<tr>
<td>$P_0$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$P_1$</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>$P_2$</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>$P_3$</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>$P_4$</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- Sequence $<P_0, P_2, P_3, P_4>$ will result in $\text{Finish}[i] = \text{true}$ for all $i$

Example (Cont.)

- $P_3$ requests an additional instance of type $C$

  Request
  - $A$ | $B$ | $C$
  - $P_3$ | 0 | 0 | 0
  - $P_4$ | 2 | 0 | 2
  - $P_5$ | 0 | 0 | 1
  - $P_6$ | 1 | 0 | 0
  - $P_7$ | 0 | 0 | 2

- State of system?
  - Can reclaim resources held by process $P_3$, but insufficient resources to fill other processes; requests
  - Deadlock exists, consisting of processes $P_1$, $P_2$, $P_3$, and $P_4$

Detection-Algorithm Usage

- When, and how often, to invoke depends on:
  - How often a deadlock is likely to occur?
  - How many processes will need to be rolled back?
    - one for each disjoint cycle
  - If detection algorithm is invoked arbitrarily,
    - There may be many cycles in the resource graph
    - And so we would not be able to tell which of the many deadlocked processes "caused" the deadlock.

Recovery from Deadlock: Process Termination

- Abort all deadlocked processes
- Abort one process at a time until the deadlock cycle is eliminated

Recovery from Deadlock: Process Termination

- In which order should we choose to abort?
  1. Priority of the process
  2. How long process has computed, and how much longer to completion
  3. Resources the process has used
  4. Resources process needs to complete
  5. How many processes will need to be terminated
  6. Is process interactive or batch?
**Recovery from Deadlock: Resource Preemption**

- **Selecting a victim** – minimize cost
- **Rollback** – return to some safe state, restart process for that state
- **Starvation** – same process may always be picked as victim, include number of rollback in cost factor

**Process Concept**

- **Process** – a program in execution; process execution must progress in sequential fashion
- **Multiple parts**
  - The program code, also called text section
  - Current activity including PC, processor registers
  - Stack containing temporary data
    - Function parameters, return addresses, local variables
  - Data section containing global variables
  - Heap containing memory dynamically allocated during run time

**Process to be run..**

- Program must be brought (from disk) into memory and placed within a process for it to be run
- CPU can access directly: Main memory and registers
- Memory unit only sees
  - A stream of addresses + read requests, or address + data and write requests

**Process to be run..**

- Register access in one CPU clock (or less)
- Main memory can take many cycles, causing a stall
- **Cache** sits between main memory and CPU registers
- Protection of memory required to ensure correct operation
**Protection Issue**
- Long term scheduler may put many process in to Ready at a time
  - Where to put?
  - How to access them?
  - Is there any efficient way to put and access?
  - How ensure safety and protection?

**Base and Limit Registers**
- A pair of registers define the logical address space
  - base and limit
- CPU must check every memory access generated in user mode to be sure it is between base and limit for that user

**Base and Limit Registers**

**Hardware Address Protection**

**Address Binding**
- Programs on disk, ready to be brought into memory to execute form an input queue
  - Without support, must be loaded into address 0000
- Inconvenient to have user process physical address always start at 0000
  - How can it not be?

**Address Binding**
- Further, addresses represented in different ways at different stages of a program’s life
  - Source code addresses usually symbolic
  - Compiled code addresses bind to relocatable addresses
    - i.e. “14 bytes from beginning of this module”
  - Linker or loader will bind relocatable addresses to absolute addresses
    - i.e. 74014
  - Each binding maps one address space to another
Binding of Instructions and Data to Memory

- Address binding can happen at three different stages
  - **Compile time**: If memory location known a priori, absolute code can be generated; must recompile code if starting location changes
  - **Load time**: Must generate relocatable code if memory location is not known at compile time
  - **Execution time**: Binding delayed until run time if the process can be moved during its execution from one memory segment to another
    - Need hardware support for address maps (e.g., base and limit registers)

Multistep Processing of a User Program

**Example**

- Program with Absolute Address
- Red marked are address
- Data are available at absolute address — LD R1 10, ...
- Call & JMP goto to Absolute Address — Call 15 and JNZ 3

**Example**

- Program with Relative Address
  - With reloadable register
  - All address need to be add with RR
- Call to Shared Lib
  - Shared Lib Register (Assumption)
  - Add at 2000
  - SR=2000
- Let program go relocated to 5000 — RelReg1=5000

Already Had Fun time

**Static Linking** and **Dynamic Linking**
Compiling multiple Files

```c
//foo.c
int foo3x(int x){
    return 3*x;
}

//bar.c
int main(){
    int x;
    x=foo3x(10);
    printf("%d",x);
    return 0;
}
```

- `$ gcc -c foo.c`
- `$ gcc -c bar.c`
- `$ gcc foo.o bar.o`
- `$ ./a.out`

Linker and Loader

- Compiler in Action...

```bash
$ gcc foo.c bar.c -o a.out
```

```
foo.c
run compiler preprocessor

bar.c
```

```
foo.s
run assembler (as)

bar.s
```

```
foo.o
linker

bar.o
```

```
a.out
= fully linked executable
```

What is Linker?

- Combines multiple relocatable object files
- Produces fully linked executable – directly loadable in memory
- How?
  - Symbol resolution – associating one symbol definition with each symbol reference
  - Relocation – relocating different sections of input relocatable files

Object files

- Types –
  - Relocatable: Requires linking to create executable
  - Executable: Loaded directly into memory for execution
  - Shared Objects: Linked dynamically, at run time or load time

Linking with Static Libraries

- Collection of concatenated object files – stored on disk in a particular format – archive
- An input to Linker
  - Referenced object files copied to executable

```bash
$ gcc -c foo.c
$ gcc -c bar.c
$ gcc foo.o bar.o
$ ./a.out
```

Connecting to Static Libraries

```c
//foo.c
int foo3x(int x){
    return 3*x;
}
```

```
int main(){
    int x; 
    x=foo3x(10);
    printf("%d",x);
    return 0;
}
```

- `$ gcc -c foo.c`
- `$ gcc -c bar.c`
- `$ gcc foo.o bar.o`
- `$ gcc -L -lfoo`
- `$ ./a.out`

Creating Static Library

```
//foo.c
int foo3x(int x){
    return 3*x;
}
```

```
int main(){
    int x; 
    x=foo3x(10);
    printf("%d",x);
    return 0;
}
```

- `$ gcc -c foo.c`
- `$ ar rcs libfoo.a foo.o`
- `$ gcc bar.c -L -lfoo`
- `$ ./a.out`
Dynamic Linking – Shared Libraries

- Addresses disadvantages of static libraries
  - Ensures one copy of text & data in memory
  - Change in shared library does not require executable to be built again
  - Loaded at run-time by dynamic linker, at arbitrary memory address, linked with programs in memory
  - On loading, dynamic linker relocates text & data of shared object

Creating Dynamic Library

```c
//foo.c
int foo3x(int x){
    return 3*x;
}
```

```c
int main(){//bar.c
    int x;
    x=foo3x(10);
    printf("%d",x);
    return 0;
}
```

- `$gcc -c -fPIC foo.c`
- `$gcc -shared -Wl,-soname,libfoo.so.1 -o libfoo.so.1 foo.o`
- `$ gcc bar.c -L -lfoo`
- `$ export LD_LIBRARY_PATH=.`
- `$./a.out`

Shared Libraries ..(Cntd)

- Linker creates `libfoo.so` (PIC) from `a.o` and `b.o`
- `a.out` – partially executable – depend on `libfoo.so`
- Dynamic linker maps shared library into program’s address space

Process in Memory:

- Stack
  - automatic (default), local
  - Initialized/uninitialized
- Data
  - Global, static, extern
  - BSS: Block Started by Symbol
  - BBS: Uninitialized Data Seg.
- Code: program instructions
- Heap: malloc, calloc

Memory for Process

- Same amount to all processes
  - Allow only N processes to be in memory
  - Size of process is limited by MemSize/N
- Variable amount of memory
  - Allow some process to be fit in the memory at a time
  - Paging & Segmentation: Improve share and reduce fragmentation
- Virtually: Allow a process to take infinite amount of memory