Multi-Level Feedback Queues and Memory Management

CS230 Tutorial 12
Thread Queues

- In our previous examples, we worked with the assumption that there was one queue of threads.
- The threads which aren't running are arranged in a queue in the order that they should run in.
- This order can be changed depending on what other threads show up.
- The order of the threads running also depends on which algorithm is being used (i.e. STCF, Round Robin, etc).
Multi-Level Feedback Queue

- Many OS’s use a multi-level queue for the scheduler
  - now we can take into account the threads using different resources (IO, etc)

- The queues have different priority levels, and threads can move between the levels as their priority changes during execution

- Each queue can use its own scheduling algorithm
  - This adds more flexibility! We can have RR in one queue and STCF in another
  - Similar to the compromise made in N-way associative cache
Multi-Level Feedback Queue - Example

Similar to what was done in class, we will define our own example of a multi-level feedback queue and observe what will happen.

Snapshot with 3 jobs:
High Priority
-------------
Q5
Q4
Q3 -> A -> B
Q2
Q1 -> C
Low Priority

Assume the following:
-- round robin in each queue
-- after one time slice in round robin, move down a priority
-- new jobs start at Q5
-- once jobs reach Q1, they stay in Q1

All jobs take 3 time slices. Also, add job D at slice 4. What job runs in each time slice, and how long does it take for all to finish?
Multi-Level Feedback Queue - Solution

Snapshot with 3 jobs:
High Priority
-----------------
Q5
Q4
Q3 -> A -> B
Q2
Q1 -> C
Low Priority

Assume the following:
-- round robin in each queue
-- after one time slice in round robin, move down a priority
-- new jobs start at Q5
-- once jobs reach Q1, they stay in Q1

All jobs take 3 time slices. Also, add job D at slice 4. What job runs in each time slice, and how long does it take for all to finish?

<table>
<thead>
<tr>
<th>Queue:</th>
<th>3 3 2 5 4 3 2 1 1 1 1 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job:</td>
<td>A B A D D D B C A B C C</td>
</tr>
<tr>
<td>Slice:</td>
<td>1 2 3 4 5 6 7 8 9 10 11 12</td>
</tr>
</tbody>
</table>
Virtual Memory

- Recall this model of a running computer from tutorial 10
- Processes each have their own address space
- No process can access memory from another process
Virtual Memory

● The addresses you use in your program are **Logical Addresses**
  ○ Also called a **Virtual Address**
  ○ Each process gets its own **Logical Address Space**
  ○ All memory of a process is inside this address space
  ○ When you `lw` or `sw` to/from a logical address the OS converts it to a **Physical Address**
  ○ A physical address corresponds to a location in actual physical memory

● How does this work?
  ○ Memory access happens all the time and takes a long time (recall caching)
  ○ We need to be fast but still safe
  ○ The **Memory Management Unit (MMU)** is hardware that does this conversion for us
Address Translation

- The are two ways of converting logical addresses to physical addresses
- Method 1: Base and Limit
  - Simple method: lets just put all the logical address spaces one after the other
  - Only works where there is much more physical memory than logical address space
  - Reserves all the memory for a process right away
  - To translate addresses: add base to the address and check limit
Address Translation

● Method 2: Paging
  ○ Separate logical address space into small chunks called **pages**
  ○ Separate physical memory into the same size chunks called **frames**
  ○ For each address space:
    ■ Keep a table that translates page numbers to frame numbers
    ■ Only attach a page to a frame when the page is first used
  ○ This stops all the wasted space (**internal fragmentation**) from Method 1
  ○ You can have more processes than you have space for address spaces in memory
    ■ Each process only consumes the physical memory it needs
  ○ Split address into page number + offset
    ■ Example with 4Kb page size:
      ● 4Kb = 4096 bytes = $2^{12} = 12$ bit offset, rest is page number
      ● Address 0x0000F345 = 0000 0000 0000 0000 1111 0011 0100 0101
        Page Number = 0xF          Offset = 0x345
Address Translation Example 1

Assume we are using the base + offset method. Process A has base 0x00004000, and Process B has base 0x0000C000. Both processes have limit 0x4000.

Where does logical address 0x1B35 in process A map to in physical memory?
Where does logical address 0x0241 in process B map to in physical memory?
Which process is using physical address 0x0000E02B?
Address Translation Example 1 Solution

Assume we are using the base + offset method. Process A has base 0x00004000, and Process B has base 0x0000C000. Both processes have limit 0x4000.

Where does logical address 0x1B35 in process A map to in physical memory?
Process A starts at 0x00004000. Add 0x00004000 + 0x1B35 = 0x00005B35

Where does logical address 0x0241 in process B map to in physical memory?
Process B starts at 0x0000C000. Add 0x0000C000 + 0x0241 = 0x0000C241

Which process is using physical address 0x0000E02B?
0x0000C000 <= 0x0000E02B
0x0000C000 + 0x4000 = 0x00010000 > 0x0000E02B
Since it is above process B’s base register and below process B’s base + limit we know it is in process B.
Address Translation Example 2

Assume we are using paging. Consider the below page tables for process A and process B. Page size is 8Kb.

A) Where is logical addresses 0x0001BAC2 in physical memory for both processes?

B) Which process is using physical address 0x000C357F and what logical address is it used as?
First we notice that the page size of 8Kb = $2^{13}$ means we have 13 bit offsets.

Where is logical addresses 0x0001BAC2 in physical memory for both processes?

<table>
<thead>
<tr>
<th>page number</th>
<th>offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000 0000 0000 0001 1011</td>
<td>1010 1100 0010</td>
</tr>
</tbody>
</table>

Therefore page number = $1101_2 = 0xD$.

Page table for process A says page 0xD is frame 0xFA = 1111 1010

<table>
<thead>
<tr>
<th>frame number</th>
<th>offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000 0000 0001 1111 0101</td>
<td>1010 1100 0010</td>
</tr>
</tbody>
</table>

= physical address 0x001F5AC2

For process B page 0xD is frame 0xD2B = 1101 0010 1011

<table>
<thead>
<tr>
<th>frame number</th>
<th>offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000 0001 1010 0101 0111</td>
<td>1010 1100 0010</td>
</tr>
</tbody>
</table>

= physical address 0x01A57AC2
Address Translation Example 2 Solutions B

Which process is using physical address 0x000C357F and what logical address is it used as?

Therefore frame number = 110 0001 = 0x61

Page table for process A says frame 0x61 is page 0xED = 1110 1101

= logical address 0x001DB57F

Process A is using physical address 0x000C357F as logical address 0x001DB57F
Reminders

- Submit a `.txt` XOR a `.pdf` for each question
  - Do not submit both for the same question!
  - You may submit a `.pdf` for one question and a `.txt` for a different question

- Make sure your diagrams and tables are clear and easy to read
  - Make sure to leave enough space

- Final exam is April 17th

Good luck!! :)

15
That's all Folks!