P2 - Preemptive Scheduling

Drew Zagieboylo
2/23/18
P1 Postmortem
P1 - Nonpreemptive

- yield()

- allow another thread to run

- w/o yield() -> single threaded behavior

```c
int thread2(int* arg) {
    minithread_fork(thread3, NULL);
    printf("Thread 2.\n");
    minithread_yield();
    return 0;
}
```
P2 - Thread Pre-emption

• How?
  • Interrupts! -> A type of Asynchronous execution

• When?
  • A timer -> uses HW clock

• What?
  • An ISR (interrupt service routine)
Interrupt Handling

• Description:
  • Register ISR for specific interrupt type
  • Enable/Disable Interrupts
  • Read Clock Value

• API:
  • minithread_clock_init(isr)
  • set_interrupt_level(level)
  • Global Variable: ‘ticks’
  • Number of clock ticks since OS start
Interrupt Handling

//proc_1

```
0xbee0 while (1){
0xbee4     x = x + 1;
0xbee8     mt_yield();
0xbeec     }
```
Interrupt Handling

// INTERRUPT!

while (1) {
    x = x + 1;
}

0xbee0
0xbec4
0xbec8
0xbec

proc_1

pc

INTERRUPT!
Interrupt Handling

```c
//proc_1

0xbe00
0xbe04
0xbe08
0xbe0c

while (1){
    x = x + 1;
}
```

```
proc_1
...

x
...
```

```
clock_handler
...

pc
```

```
...
```

pc
```
```
```
Interrupt Handling

clock_handler
{
  ...
  //pick next thread
  //mt_switch to next thread
  //re-enable interrupts

  proc_1
  ...
  x
  ...
  ...
}
Interrupt Safety

- Critical Section -> some need to be interrupt safe
- Don’t forget to re-enable interrupts when done!
- When ISR starts:
  - Interrupts must be disabled
  - DON’T block (sema_P) while handling interrupts
- Semaphore updates must be interrupt-safe
Semaphore

semaphore_P(sema) {
    sema->count--;
    if (count < 0) {
        queue_append(sema->q, minithread_self());
        minithread_stop();
    }
}

These lines must happen atomically ->
in Port OS this requires interrupt safety
Alarms!

- Description:
  - Asynchronous execution
  - Execute some function at a future time
  - Can ‘cancel’ them
  - *Interrupt Safety*

- API:
  - `alarm_register(delay, func)`
  - `alarm_deregister(alarm)`
Alarms!

- Every clock tick
  - Check alarms -> execute any that are due to execute
  - Must run in $O(n)$, $n =$ number of ready alarms
  - **NOT** $O(r)$, $r =$ number of registered alarms
  - (You may need to modify your queue API)
Alarms

• You’ll implement ‘minithread_sleep_with_timeout’ as an exercise
• Deschedules thread for a fixed amount of time
• Should be a very short bit of code :)

Scheduling Algorithm

• Need a way to pick the next thread to run
• (Do this after everything else works)
• As of P1 - FIFO
Multilevel Feedback Queue

- **High Priority (Low Level Num)**
  Quick Tasks -> need low latency

- Usually I/O heavy

- **Low Priority (High Level Num)**
  Need more CPU time -> needs more throughput

- computationally heavy
Multilevel Feedback Queue

- High Priority (Low Level Num)
  - Give more CPU time overall
  - Less CPU time per task

- Low Priority (High Level Num)
  - Less CPU time overall
  - More CPU time per task
# Multilevel Feedback Queue

<table>
<thead>
<tr>
<th>Time Allocated Per Thread</th>
<th>Time Allocated Per Queue</th>
<th>Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2^0$</td>
<td>5$t$</td>
<td>0</td>
</tr>
<tr>
<td>$2^1$</td>
<td>2.5$t$</td>
<td>1</td>
</tr>
<tr>
<td>$2^2$</td>
<td>1.5$t$</td>
<td>2</td>
</tr>
<tr>
<td>$2^3$</td>
<td>$t$</td>
<td>3</td>
</tr>
</tbody>
</table>
Multilevel Feedback Queue

<table>
<thead>
<tr>
<th>Time Allocated Per Thread</th>
<th>Time Allocated Per Queue</th>
<th>Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2^0$</td>
<td>5t</td>
<td>0</td>
</tr>
<tr>
<td>$2^1$</td>
<td>2.5t</td>
<td>1</td>
</tr>
<tr>
<td>$2^2$</td>
<td>1.5t</td>
<td>2</td>
</tr>
<tr>
<td>$2^3$</td>
<td>t</td>
<td>3</td>
</tr>
</tbody>
</table>

Start a new Thread
# Multilevel Feedback Queue

<table>
<thead>
<tr>
<th>Time Allocated Per Thread</th>
<th>Time Allocated Per Queue</th>
<th>Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2^0$</td>
<td>$5t$</td>
<td>0</td>
</tr>
<tr>
<td>$2^1$</td>
<td>$2.5t$</td>
<td>1</td>
</tr>
<tr>
<td>$2^2$</td>
<td>$1.5t$</td>
<td>2</td>
</tr>
<tr>
<td>$2^3$</td>
<td>$t$</td>
<td>3</td>
</tr>
</tbody>
</table>

After 1 tick, thread still executing
Multilevel Feedback Queue

<table>
<thead>
<tr>
<th>Time Allocated Per Thread</th>
<th>Time Allocated Per Queue</th>
<th>Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2^0$</td>
<td>5t</td>
<td>0</td>
</tr>
<tr>
<td>$2^1$</td>
<td>2.5t</td>
<td>1</td>
</tr>
<tr>
<td>$2^2$</td>
<td>1.5t</td>
<td>2</td>
</tr>
<tr>
<td>$2^3$</td>
<td>t</td>
<td>3</td>
</tr>
</tbody>
</table>

Demote thread to LVL 1
### Multilevel Feedback Queue

<table>
<thead>
<tr>
<th>Time Allocated Per Thread</th>
<th>Time Allocated Per Queue</th>
<th>Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2^0$</td>
<td>5t</td>
<td>0</td>
</tr>
<tr>
<td>$2^1$</td>
<td>1.5t</td>
<td>1</td>
</tr>
<tr>
<td>$2^2$</td>
<td>t</td>
<td>2</td>
</tr>
<tr>
<td>$2^3$</td>
<td></td>
<td>3</td>
</tr>
</tbody>
</table>

Pick another thread from LVL 0 to run.
Multilevel Feedback Queue

<table>
<thead>
<tr>
<th>Time Allocated Per Thread</th>
<th>Time Allocated Per Queue</th>
<th>Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2^0$</td>
<td>5t</td>
<td>0</td>
</tr>
<tr>
<td>$2^1$</td>
<td>1.5t</td>
<td>1</td>
</tr>
<tr>
<td>$2^2$</td>
<td>t</td>
<td>2</td>
</tr>
<tr>
<td>$2^3$</td>
<td></td>
<td>3</td>
</tr>
</tbody>
</table>

Eventually...
Pick a thread from LVL 1 Instead