CRPD Independence for multiple process execution

Jan Staschulat and Rolf Ernst
Institute of Computer and Communication Network Engineering (IDA)
Technical University of Braunschweig,
D-38106 Braunschweig, Germany
{staschulat,ernst}@ida.ing.tu-bs.de

1 Problem Statement

Current approaches for cache related preemption delay analysis (CRPD) [1], [2], [3] assume an empty cache at process start. But often a process is activated frequently depending on the scheduling algorithms. However, in real time embedded systems application profit only from a cache, if some cache lines survive from earlier process executions [4].

Between two activations higher and lower priority processes can execute. This proof shows that the order and the frequency of these intermediate process executions does not change the CRPD for the second process execution.

2 Proof

We show the independence of the CRPD for a n-way associative instruction cache in two steps.

\[ CB_s(\tau_i, \tau_k, \tau_l) = CB_s(\tau_i, \tau_l) \]  \hspace{1cm} (1)

\[ CB_s(\tau_i, \tau_l, \tau_k) = CB_s(\tau_i, \tau_k, \tau_l) \]  \hspace{1cm} (2)

Where \( CB_s(\tau_i, \tau_k, \tau_l) \) denotes the number and the order of cache blocks in cache set \( s \) of \( \tau_i \) after the execution of processes \( \tau_i, \tau_k, \tau_l \). Equation 1 says that multiple executions of \( \tau_k \) can be reduced to a single executing \( \tau_k \). Equation 2 says that the remaining cache blocks are independent of the execution order. Thus any execution sequence of arbitrary order and frequency can transformed to a canonical sequence of process executions without repetitions by applying eq. 1 and eq. 2. If this is proven we have shown that the contents and the order of cache set \( s \) for \( \tau_i \) is the same for any
order and frequency of intermediate processes. Hence the CRPD at a second activation of $\tau_i$ is independent of the order and frequency of intermediate processes.

We assume an associative instruction cache with LRU replacement strategy. It suffices to show both equations for a cache set. If they hold for a cache set then they are true for the entire cache, because the cache behaviour is independent of its sets. An n-way associative cache set has n cache blocks.

When a cache block $B_i$ is loaded to a full cache set the LRU strategy replaces the least recently used cache block by $B_i$. If cache block $B_i$ is already in the cache set and requested again by the CPU, then only the order of all cache blocks is changed. Suppose that a cache set holds n cache blocks. Further let $K$, $L$, and $I$ denote the number of cache blocks of process $\tau_k$, $\tau_i$, and $\tau_i$ respectively. Let $M$ be the total number of new cache blocks that are mapped to a cache set. This can be $M = L$ for eq. 1 or $M = L + K$ for eq. 2. We consider three cases:

1. $M \geq n$. The number of new cache blocks is greater than the associativity. This means particularly that all $I$ cache blocks of $\tau_i$ are replaced.

2. $M \leq n - I$. All new cache blocks fit in the cache set and none of the $I$ cache blocks of process $\tau_i$ are replaced.

3. $n - I < M < n$. The number of new cache blocks is larger then the vacant positions of the cache set, but smaller then the total number of cache blocks. In this case, $I + M - n$ cache blocks of $\tau_i$ are replaced.

First we show that eq. 1 holds. We set $M = K$. For case 1. the number of remaining cache blocks of process $\tau_i$ is zero, if process $\tau_k$ is executed once or twice. For case 2. no cache blocks of $\tau_i$ are replaced after one execution of $\tau_k$. Also during the second execution of $\tau_k$ no cache blocks are replaced.

For case 3 we have to show

$$\forall m' \in S(\tau_i, \tau_k, \tau_k) \rightarrow m' \in S(\tau_i, \tau_k)$$  (3)

$$\forall m' \in S(\tau_i, \tau_k) \rightarrow m' \in S(\tau_i, \tau_k, \tau_k)$$  (4)

Eq 3 is trivial. Suppose $\exists m' \in S(\tau_i, \tau_k, \tau_k)$ and $m' \notin S(\tau_i, \tau_k)$. During execution of $\tau_k$ no cache blocks of $\tau_i$ are loaded, consequently $m'$ cannot be in the cache set. So $m' \notin S(\tau_i, \tau_k, \tau_k)$ which is a contradiction. For eq. 4 suppose $m' \in S(\tau_i, \tau_k)$ arbitrarily, which means after the execution of $\tau_k$ $m'$ is still in the cache set. During the second execution of $\tau_k$ the access to all its cache blocks $m_{k1}, \ldots, m_{kK}$ are all cache hits, because $M < n$. So only
a reordering according to LRU strategy can occur. Particularly no cache block of $\tau_i$ is replaced, so $m' \in S(\tau_i, \tau_k, \tau_k)$. We have now shown that for all cases eq. 1 holds.

Equation 2 trivially holds for case 1 and 2. Here we set $M = K + L$. In case 1 all $\tau_i$ cache blocks and in case 2 no cache blocks are replaced. For case 3 assume $m' \in S(\tau_i, \tau_k, \tau_l)$. So the least $K + L - n$ cache blocks are replaced. No cache blocks of $\tau_k$ can be replaced by $\tau_l$ since $K + L < n$. Consequently all $K + L - n$ cache blocks are replaced from $\tau_i$. Similarly it follows from the LRU strategy that the $L + K - n$ cache blocks being replaced during execution of $\tau_l$ and $\tau_k$ must be from $\tau_i$. The order of the I cache blocks of $\tau_i$ cannot change during execution of $\tau_k$ and $\tau_l$ and execution of $\tau_i$ and $\tau_k$ because the I cache blocks are loaded before the execution of $\tau_l$ and $\tau_k$. So in both execution orders the same cache blocks of $\tau_i$ are replaced. This proves equation 2 for case 3.

**References**


