Process Interactions and Blocking

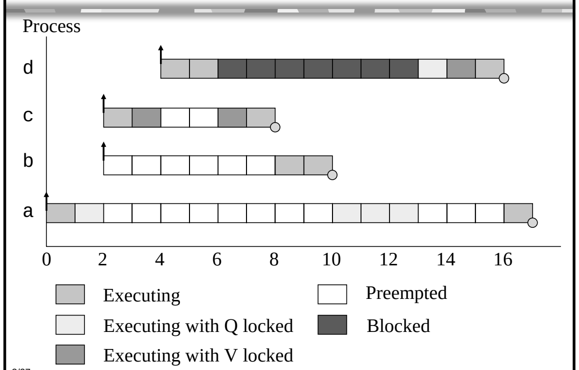
- If a process is suspended waiting for a lower-priority process to complete some required computation then the priority model is, in some sense, being undermined
- It is said to suffer priority inversion
- If a process is waiting for a lower-priority process, it is said to be blocked

Priority Inversion – 1

- To illustrate an extreme example of priority inversion, consider the executions of four periodic processes: a, b, c and d; and two resources: `Q` and `V`

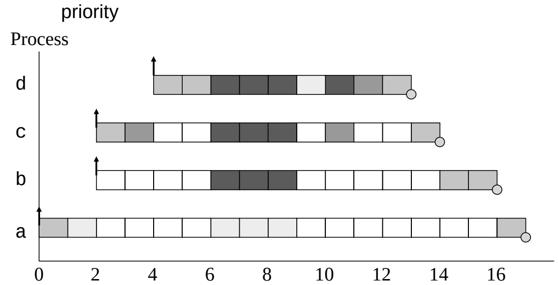
Process	Priority	Execution Sequence	Release Time
a	1	EQQQQE	0
b	2	EE	2
c	3	EVVE	2
d	4	EEQVE	4

Priority Inversion – 2



Priority Inheritance – 3

- If process p is blocking process q, then q runs with p's priority



Calculating Blocking

- If a process has `m` critical sections that can lead to it being blocked then the maximum number of times it can be blocked is `m`
- If `B` is the maximum blocking time and `K` is the number of critical sections, the process `i` has an upper bound on its blocking given by:

$$B_i = \sum_{k=1}^{K} usage(k,i)C(k)$$

- Where *usage(k,i) = 1* if resource *k* is used by at least one process with priority less than P_i, otherwise it evaluates to *0*

Response Time and Blocking

$$R_i = C_i + B_i + I_i$$

$$R_i = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

$$w_i^{n+1} = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i^n}{T_j} \right\rceil C_j$$

Priority Ceiling Protocols

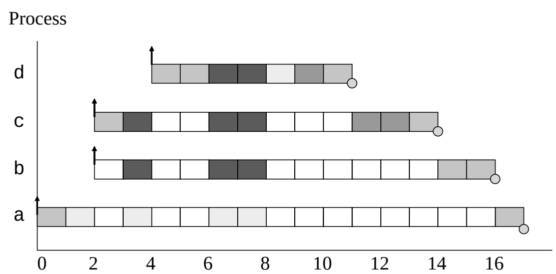
- It takes on two forms
  - Original ceiling priority protocol
  - Immediate ceiling priority protocol
- Owing to them, on a single processor:
  - A high-priority process can be blocked by lower-priority processes at most once during its execution
  - Deadlocks are prevented
  - Transitive blocking is prevented
  - Mutual exclusive access to resources is ensured by the protocol itself

Original Ceiling Priority Protocol

- Each process has a static default priority assigned (perhaps by the deadline monotonic scheme)
- Each resource has a static ceiling value defined, this is the maximum priority of the processes that use it
- A process has a dynamic priority that is the maximum of its own static priority and any it inherits due to it blocking higher-priority processes
- A process can only lock a resource if its dynamic priority is higher than the ceiling of any currently locked resource (excluding any that it has already locked itself)

$$B_i = \max_{k=1}^{k} usage(k,i)C(k)$$

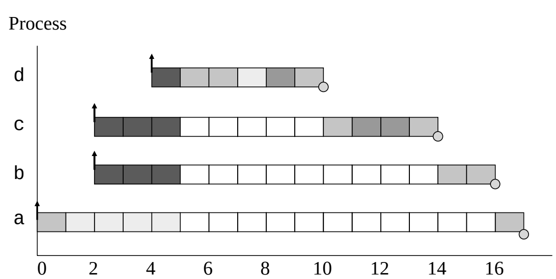
OCPP Inheritance



Immediate Ceiling Priority Protocol

- Each process has a static default priority assigned (perhaps by the deadline monotonic scheme)
- Each resource has a static ceiling value defined, this is the maximum priority of the processes that use it
- A process has a dynamic priority that is the maximum of its own static priority and the ceiling values of any resources it has locked
- As a consequence, a process will only suffer a block at the very beginning of its execution
- Once the process starts actually executing, all the resources it needs must be free; if they were not, then some process would have an equal or higher priority and the process' execution would be postponed

ICPP Inheritance



OCPP versus ICPP

- Although the worst-case behaviour of the two ceiling schemes is identical (from a scheduling view point), there are some points of difference:
  - ICPP is easier to implement than the original (OCPP) as blocking relationships need not be monitored
  - ICPP leads to less context switches as blocking is prior to first execution
  - ICPP requires more priority movements as this happens with all resource usage
  - OCPP changes priority only if an actual block has occurred
- Note that ICPP is called Priority Protect Protocol in POSIX and Priority Ceiling Emulation in Real-Time Java

An Extendible Process Model

- What the model allows so far:
  - Deadlines can be less than period (D<T)
  - Sporadic and aperiodic processes, as well as periodic processes, can be supported
  - Process interactions are possible, with the resulting blocking being factored into the response time equations

Extensions

- Cooperative Scheduling
- Release Jitter
- Arbitrary Deadlines
- Fault Tolerance
- Offsets
- Optimal Priority Assignment

Cooperative Scheduling – 1

- True preemptive behaviour is not always acceptable for safety-critical systems
- Cooperative or deferred preemption splits processes into slots
- Mutual exclusion is via non-preemption
- The use of deferred preemption has two important advantages
  - It increases the schedulability of the system, and it can lead to lower values of `C`
  - With deferred preemption, no interference can occur during the last slot of execution

Cooperative Scheduling – 2

- Let the execution time of the final block (slot) be F_i

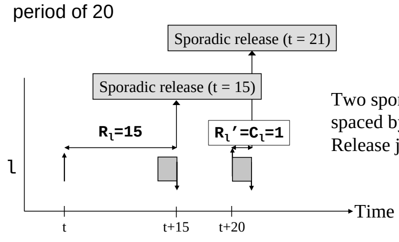
$$w_i^{n+1} = B_{MAX} + C_i - F_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i^n}{T_j} \right\rceil C_j$$

- When this converges that is, $w_i^n = w_i^{n+1}$, the response time is given by:

$$R_i = w_i^n + F_i$$

Release Jitter – 1

- A key issue for distributed systems
- Consider the release of a sporadic process on a different processor by a periodic process, `l`, with a period of 20



Two sporadic releases spaced by 21-15 = 6 ! Release jitter $J_s = R_l$

Release Jitter – 2

- Sporadic process `s` released at `0, T-J, 2T-J, 3T-J`
- Examination of the derivation of the schedulability equation implies that process `i` will suffer
  - one interference from process `s` if $R_i \in [0, T-J)$
  - two interferences if $R_i \in [T-J, 2T-J)$
  - three interference if $R_i \in [2T-J, 3T-J)$
- This can be represented in the response time equations

$$R_i = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i + J_j}{T_j} \right\rceil C_j$$

- If response time is to be measured relative to the real release time then the jitter value must be added

$$R_i^{periodic} = R_i + J_i$$

Arbitrary Deadlines

- To cater for situations where D (and hence potentially R) > T

$$w_i^{n+1}(q) = B_i + (q+1)C_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i^n(q)}{T_j} \right\rceil C_j$$

$$R_i(q) = w_i^n(q) - qT_i$$

- The number of releases is bounded by the lowest value of q for which the following relation is true: $R_i(q) \le T_i$
- The worst-case response time is then the maximum value found for each q: $R_i = \max_{q=0,1,2,\ldots} R_i(q)$

Arbitrary Deadlines

- When formulation is combined with the effect of release jitter, two alterations to the above analysis must be made
- First, the interference factor must be increased if any higher priority processes suffers release jitter:

$$w_i^{n+1}(q) = B_i + (q+1)C_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i^n(q) + J_j}{T_j} \right\rceil C_j$$

- The other change involves the process itself. If it can suffer release jitter then two consecutive windows could overlap if response time plus jitter is greater than period

$$R_i(q) = w_i^n(q) - qT_i + J_i$$

Fault Tolerance

- Fault tolerance via either forward or backward error recovery always results in extra computation
- This could be an exception handler or a recovery block.
- In a real-time fault-tolerant system, deadlines should still be met even when a certain level of faults occur
- This level of fault tolerance is known as the fault model
- If the extra computation time that results from an error in process `i` is C_i^f

$$R_i = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j + \max_{k \in hep(i)} C_k^f$$

- where `hep(i)` is set of processes with priority equal to or higher than `i`

Fault Tolerance

- If F is the number of faults allowed

$$R_i = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j + \max_{k \in hep(i)} FC_k^f$$

- If there is a minimum arrival interval T_f

$$R_i = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j + \max_{k \in hep(i)} \left(\left\lceil \frac{R_i}{T_f} \right\rceil C_k^f \right)$$

Offsets

- So far assumed all processes share a common release time (critical instant)

Process	T	D	C	R	U=0.9
a	8	5	4	4	
b	20	10	4	8	Deadline miss!
c	20	12	4	16	

- With offsets

Process	T	D	C	O	R
a	8	5	4	0	4
b	20	10	4	0	8
c	20	12	4	10	**8**

Arbitrary offsets are not amenable to analysis!

Non-Optimal Analysis – 1

- In most realistic systems, process periods are not arbitrary but are likely to be related to one another
- As in the example just illustrated, two processes have a common period. In these situations it is ease to give one an offset (of `T/2`) and to analyze the resulting system using a transformation technique that removes the offset — and, hence, critical instant analysis applies
- In the example, processes b and c (having the offset of 10) are replaced by a single notional process with period 10, computation time 4, deadline 10 but no offset

Non-Optimal Analysis – 2

- This notional process has two important properties
  - If it is schedulable (when sharing a critical instant with all other processes) then the two real processes will meet their deadlines when one is given the half period offset
  - If all lower priority processes are schedulable when suffering interference from the notional process (and all other high-priority processes) then they will remain schedulable when the notional process is replaced by the two real processes (one with the offset)
- These properties follow from the observation that the notional process always has no less CPU utilization than the two real processes

Process	T	D	C	O	R	U=0.9
a	8	5	4	0	4	
n	10	10	4	0	**8**	

Notional Process Parameters

$$T_n = \frac{T_a}{2} = \frac{T_b}{2}$$
$$C_n = Max(C_a, C_b)$$
$$D_n = Min(D_a, D_b)$$
$$P_n = Max(P_a, P_b)$$

Can be extended to more than two processes

Priority Assignment

- **Theorem**: If process p is assigned the lowest priority and is feasible then, if a feasible priority ordering exists for the complete process set, an ordering exists with process p assigned the lowest priority

```
procedure Assign_Pri (Set : in out Process_Set;
                      N   : Natural; -- number of processes
                      OK  : out Boolean) is
begin
  for K in 1..N loop
    for Next in K..N loop
      Swap(Set, K, Next);
      Process_Test(Set, K, OK); -- is process K feasible now?
      exit when OK;
    end loop;
    exit when not OK; -- failed to find a schedulable process
  end loop;
end Assign_Pri;
```