```
Monitor PriorityQueueAging

Const
Mmax : integer; // max consecutive high-priority jobs

Var
busy : boolean; // true if a job is being served
waitingHigh : integer; // number of waiting high jobs
waitingLow : integer; // number of waiting low jobs
servedHighCount : integer; // consecutive high jobs
served
canHigh, canLow : condition;

Procedure RequestHigh;
Begin
waitingHigh := waitingHigh + 1;

While busy do
canHigh.wait;

// this high job will be served now
waitingHigh := waitingHigh - 1;
busy := true;
End;

Procedure ReleaseHigh;
Begin
busy := false;
servedHighCount := servedHighCount + 1;

// Aging: after Mmax highs and if a low is waiting, give
turn to low
If (servedHighCount >= Mmax) AND (waitingLow > 0)
then
Begin
servedHighCount := 0;
canLow.signal;
End
else
// otherwise prefer high jobs if any, else lows
If waitingHigh > 0 then
canHigh.signal
else if waitingLow > 0 then
canLow.signal;
End;

Procedure RequestLow;
Begin
waitingLow := waitingLow + 1;

While busy
OR (waitingHigh > 0) AND (servedHighCount < Mmax) do
canLow.wait;

// this low job will be served now
waitingLow := waitingLow - 1;
busy := true;
servedHighCount := 0; // reset after a low job
End;

Procedure ReleaseLow;
Begin
busy := false;

// After a low job, again prefer highs if any
If waitingHigh > 0 then
canHigh.signal
else if waitingLow > 0 then
canLow.signal;
End;

Begin
busy := false;
waitingHigh := 0;
waitingLow := 0;
servedHighCount := 0;
End;

===end monitor

Process P_High_i;
Begin
Call PriorityQueueAging.RequestHigh;
ServeHighJob(); // critical section
Call PriorityQueueAging.ReleaseHigh;
End;
```

```
Process P_Low_j;
Begin
Call PriorityQueueAging.RequestLow;
ServeLowJob(); // critical section
Call PriorityQueueAging.ReleaseLow;
End.
```