

# ECS 235B Module 25

## Availability Models (*con't*)

# State-Based Model (Millen)

- Unlike constraint-based model, allows a maximum waiting time to be specified
- Based on resource allocation system, denial of service base that enforces its policies

# Resource Allocation System Model

- $R$  set of resource types
- For each  $r \in R$ , number of resource units (capacity,  $c(r)$ ) is constant; a process can hold a unit for a maximum holding time  $m(r)$
- $P$  set of processes
- For each  $p \in P$ , state is *running* or *sleeping*
  - When allocated a resource, process is running
  - Multiple process can be in running state simultaneously
  - Each  $p$  has upper bound it can be in running state before being interrupted, if only by CPU quantum  $q$
  - Example: if CPU considered a resource,  $m(\text{CPU}) = q$

# Allocation Matrix

- Rows represent processes; columns represent resources
  - $A: P \times R \rightarrow \mathbb{N}$  is matrix
  - For  $p \in P, r \in R, A_p(r)$  is number of resource units of type  $r$  acquired by  $p$
  - As at most  $c(r)$  of resource type  $r$  exist, at most that many can be allocated at any time

R1: The system cannot allocate more instances of a resource type than it has:

$$(\forall r \in R)[\sum_{p \in P} A_p(r) \leq c(r)]$$

# More About Resources

- $T: P \rightarrow \mathbb{N}$  is system time when resource assignment was last changed
  - Think of it as a time vector, each element belonging to one process
- $Q^S: P \times R \rightarrow \mathbb{N}$  is matrix of required resources for each process, *not including the resources it already holds*
  - So  $Q^S_p(r)$  means the number of units of resource type  $r$  that process  $p$  may need to complete
- $Q^T: P \times R \rightarrow \mathbb{N}$  is matrix of how much longer each process  $p$  needs the units of resource  $r$
- Predicates  $running(p)$  true if  $p$  is in running state;  $asleep(p)$  true otherwise

R2: A currently running process must not require additional resources to run

$$running(p) \Rightarrow (\forall r \in R)[Q^S_p(r) = 0]$$

# States, State Transitions

- Current state of system is  $(A, T, Q^S, Q^T)$
- State transition  $(A, T, Q^S, Q^T) \rightarrow (A', T', Q^{S'}, Q^{T'})$ 
  - We only care about transitions due to allocation, deallocation of resources
- Three relevant types of transitions
  - *Deactivation transition*:  $running(p) \rightarrow asleep'(p)$ ; process stops execution
  - *Activation transition*:  $asleep(p) \rightarrow running'(p)$ ; process starts or resumes execution
  - *Reallocation transition*: transition in which  $p$  has resource allocation changed; can only occur when  $asleep(p)$

# Constraints

R3: Resource allocation does not affect allocations of a running process:

$$(running(p) \wedge running'(p)) \Rightarrow (A_p' = A_p)$$

R4:  $T(p)$  changes only when resource allocation of  $p$  changes:

$$(A_p'(CPU) = A_p(CPU)) \Rightarrow (T'(p) = T(p))$$

R5: Updates in time vector increase value of element being updated:

$$(A_p'(CPU) \neq A_p(CPU)) \Rightarrow (T'(p) > T(p))$$

# Constraints

R6: When  $p$  reallocated resources, allocation matrix updated before  $p$  resumes execution:

$$asleep(p) \Rightarrow Q_p^S' = Q_p^S + A_p - A_p'$$

R7: When a process is not running, the time it needs resources does not change:

$$asleep(p) \Rightarrow Q_p^T' = Q_p^T$$

R8: when a process ceases to execute, the only resource it *must* surrender is the CPU:

$$(running(p) \wedge asleep'(p)) \Rightarrow A_p'(r) = A_p(r) - 1 \quad \text{if } r = \text{CPU}$$

$$(running(p) \wedge asleep'(p)) \Rightarrow A_p'(r) = A_p(r) \quad \text{otherwise}$$



# Resource Allocation System

- A system in a state  $(A, T, Q^S, Q^T)$  such that:
  - State satisfies constraints R1, R2
  - All state transitions constrained to meet R3-R8

# Denial of Service Protection Base (DPB)

- A mechanism that is tamperproof, cannot be prevented from operating, and guarantees authorized access to resources it controls
- Four parts:
  - Resource allocation system (see earlier)
  - Resource monitor
  - Waiting time policy
  - User agreement (see earlier); constraints apply to changes in allocation when process transitions from *running(p)* to *asleep(p)*

# Resource Monitor

- Controls allocation, deallocation of resources and the timing
- $Q_p^S$  is *feasible* if  $(\forall i)[Q_p^S(r_i) + A_p(r_i) \leq c(r_i)] \wedge Q_p^S(\text{CPU}) \leq 1$ 
  - If the total number of resources it will be allocated will always be no more than the capacity of that resource, and no more than 1 CPU is requested
- $T_p$  is *feasible* if  $(\forall i)[T_p(r_i) \leq \max(r_i)]$ 
  - Here,  $\max(r_i)$  max time a process must wait for its needed allocation of units of resource type  $i$

# Waiting Time Policy

- Let  $\sigma = (A, T, Q^S, Q^T)$

- Example finite waiting time policy:

$$(\forall p, \sigma)(\exists \sigma')[\text{running}'(p) \wedge (T'(p) \geq T(p))]$$

- For every process and state, there is a future state in which  $p$  is executing and has been allocated resources

- Example maximum waiting time policy:

$$(\exists M)(\forall p, \sigma)(\exists \sigma')[\text{running}'(p) \wedge (0 < T'(p) - T(p) \leq M)]$$

- There is an upper bound  $M$  to how long it takes every process to reach a future state in which it is executing and has been allocated resources

# Two Additional Constraints

In addition to all these, a DPB must satisfy these constraints:

1. Each process satisfying user agreement constraints will progress in a way that satisfies the waiting time policy
2. No resource other than the CPU is deallocated from a process unless that resource is no longer needed

$$(\forall i)[r_i \neq \text{CPU} \wedge A_p(r_i) \neq 0 \wedge A_p'(r_i) = 0] \Rightarrow Q_p^T(r_i) = 0$$

# Example: DPB

- Assume system has 1 CPU
- Assume maximum waiting time policy in place
- 3 parts to user agreement:
  - $Q_p^S, T_p$  are *feasible*
  - Process in running state executes for a minimum amount of time before it transitions to a non-running state
  - If process requires resource type, and enters a non-running state, the time it needs the resource for is decreased by the amount of time it was in the previous running state; that is,

$$Q_p^T \neq \mathbf{0} \wedge \text{running}(p) \wedge \text{asleep}'(p) \Rightarrow (\forall r \in R)[Q_p^T(r) \leq \max(0, \max_r Q_p^T(r) - (T'(p) - T(p)))]$$

# Example: System

- $n$  processes, round robin scheduler with quantum  $q$
- Initially no process has any resources
- Resource monitor selects process  $p$  to give resources to
  - $p$  executes until  $Q_p^T = \mathbf{0}$  or monitor concludes  $Q_p^S$  or  $T_p$  is not feasible
- Goal: show there will be no denial of service in this system because
  - a) no resource  $r_i$  is deallocated from  $p$  for which  $Q_p^S$  is feasible until  $Q_p^T = \mathbf{0}$ ;  
and
  - b) there is a maximum time for each round robin cycle

# Claim (a)

- Before  $p$  selected, no process has any resources allocated to it
  - So next process with  $Q_p^S$  and  $T_p$  feasible is selected
  - It runs until it enters the *asleep* state or  $q$ , whichever is shorter
  - If in *asleep* state, process is done
  - If  $q$ , monitor gives  $p$  another quantum of running time; this repeats until  $Q_p^T = 0$ , and then  $p$  needs no more resources
- Let  $m(r)$  be maximum time any process will hold resources of type  $r$ 
  - Let  $M(r) = \max_r m(r)$
- As  $Q_p^S$  and  $T_p$  feasible,  $M$  upper bound for all elements of  $Q_p^T$ 
  - $d = \min(q, \text{minimum time before } p \text{ transitions to } \textit{asleep} \text{ state})$ ; exists because a process in running state executes for a minimum amount of time before it transitions to a non-running state



# Claim (a) (*con't*)

- As  $Q_p^S$  and  $T_p$  feasible,  $M$  upper bound for all elements of  $Q_p^T$
- $d = \min(q, \text{minimum time before } p \text{ transitions to } \textit{asleep} \text{ state})$ 
  - Exists because a process in running state executes for a minimum amount of time before it transitions to a non-running state
- At end of each quantum,  $m'(r) = m(r) - d$ 
  - By third part of user agreement
- So after  $\text{floor}(M/d + 1)$  quanta,  $Q_p^T = \mathbf{0}$ 
  - So no resources deallocated until  $(\forall i) Q_p^T(r_i) = 0$

# Claim (b)

- $t_a$  is time between resource monitor beginning cycle and when it has allocated required resources to  $p$
- Resource monitor then allocates CPU resource to  $p$ ; call this time  $t_{\text{CPU}}$ 
  - Done between each quantum
- When  $p$  completes, all its resources deallocated; this takes time  $t_d$
- As  $Q_p^S$  and  $T_p$  feasible, time needed to run  $p$ , including time to deallocate all resources, is:

$$t_a + \text{floor}(M/d + 1)(q + t_{\text{CPU}}) + t_d$$

- So for  $n$  processes, maximum time cycle will take is  $n$  times this
- Thus, there is a maximum time for each round robin cycle

# Quiz

True or false: the system in the example uses a round robin scheduling technique. Would it be vulnerable to a denial of service attack if the scheduling algorithm were shortest job first?