ECS 235B Module 25
Constraint-Based Availability Models
Goals

• Ensure a resource can be accessed in a timely fashion
  • Called “quality of service”
  • “Timely fashion” depends on nature of resource, the goals of using it

• Closely related to safety and liveness
  • Safety: resource does not perform correctly the functions that client is expecting
  • Liveness: resource cannot be accessed
Key Difference

- Mechanisms to support availability in general
  - Lack of availability assumes average case, follows a statistical model
- Mechanisms to support availability as security requirement
  - Lack of availability assumes worst case, adversary deliberately makes resource unavailable
  - Failures are non-random, may not conform to any useful statistical model
Deadlock

• A state in which some set of processes block each waiting for another process in set to take some action
  • *Mutual exclusion*: resource not shared
  • *Hold and wait*: process must hold resource and block, waiting other needed resources to become available
  • *No preemption*: resource being held cannot be released
  • *Circular wait*: set of entities holding resources such that each process waiting for another process in set to release resources

• Usually not due to an attack
Approaches to Solving Deadlocks

• **Prevention**: prevent 1 of the 4 conditions from holding
  • Do not acquire resources until all needed ones are available
  • When needing a new resource, release all held

• **Avoidance**: ensure process stays in state where deadlock cannot occur
  • *Safe state*: deadlock can not occur
  • *Unsafe state*: may lead to state in which deadlock can occur

• **Detection**: allow deadlocks to occur, but detect and recover
Denial of Service

• Occurs when a group of authorized users of a service make that service unavailable to a (disjoint) group of authorized users for a period of time exceeding a defined maximum waiting time
  • First “group of authorized users” here is group of users with access to service, whether or not the security policy grants them access
  • Often abbreviated “DoS” or “DOS”

• Assumes that, in the absence of other processes, there are enough resources
  • Otherwise problem is not solvable unless more resources created
  • Inadequate resources is another type of problem
Components of DoS Model

• *Waiting time policy*: controls the time between a process requesting a resource and being allocated that resource
  • Denial of service occurs when this waiting time exceeded
  • Amount of time depends on environment, goals

• *User agreement*: establishes constraints that process must meet in order to access resource
  • Here, “user” means a process
  • These ensure a process will receive service within the waiting time
Constraint-Based Model (Yu-Gligor)

- Framed in terms of users accessing a server for some services
- *User agreement*: describes properties that users of servers must meet
- *Finite waiting time policy*: ensures no user is excluded from using resource
User Agreement

• Set of constraints designed to prevent denial of service
• $S_{seq}$ sequence of all possible invocations of a service
• $U_{seq}$ set of sequences of all possible invocations by a user
• $U_{li,seq} \subseteq U_{seq}$ that user $U_i$ can invoke
  • $C$ set of operations $U_i$ can perform to consume service
  • $P$ set of operations to produce service user $U_i$ consumes
  • $p < c$ means operation $p \in P$ must precede operation $c \in C$
  • $A_i$ set of operations allowed for user $U_i$
  • $R_i$ set of relations between every pair of allowed operations for $U_i$
Example

Mutually exclusive resource

• $C = \{ \text{acquire} \}$
• $P = \{ \text{release} \}$
• For $p_1, p_2, A_i = \{ \text{acquire}_i, \text{release}_i \}$ for $i = 1, 2$
• For $p_1, p_2, R_i = \{ \text{acquire}_i < \text{release}_i \}$ for $i = 1, 2$
Sequences of Operations

- $U_i(k)$ initial subsequence of $U_i$ of length $k$
  - $n_o(U_i(k))$ number of times operation $o$ occurs in $U_i(k)$
- $U_i(k)$ safe if the following 2 conditions hold:
  - if $o \in U_i,_{seq}$ then $o \in A_i$; and
    - That is, if $U_i$ executes $o$, it must be an allowed operation for $U_i$
  - for all $k$, if $(o < o') \in R_i$, then $n_o(U_i(k)) \geq n_{o'}(U_i(k))$
    - That is, if one operation precedes another, the first one must occur more times than the second
Resources of Services

• \( s \in S_{seq} \) possible sequence of invocations of services

• \( s \) blocks on condition \( c \)
  • May be waiting for service to become available, or processing some response, etc.

• \( o_i^*(c) \) represents operation \( o_i \) blocked, waiting for \( c \) to become true
  • When execution results, \( o_i(c) \) represents operation
  • Note that when \( c \) becomes true, \( o_i^*(c) \) may not resume immediately
Resources of Services

- \( s(0) \) initial subsequence of \( s \) up to operation \( o_i^*(c) \)
- \( s(k) \) subsequence of operations between \((k-1)^{st}, k^{th}\) time \( c \) becomes true after \( o_i^*(c) \)
- \( o_i^*(c) \) \( \rightarrow^{s(k)} o_i(c) \): \( o_i \) blocks waiting on \( c \) at end of \( s(0) \), resumes operation at end of \( s(k) \)
- \( S_{seq} \) live if for every \( o_i^*(c) \) there is a set of subsequences \( s(0), ..., s(k) \) such that it is initial subsequence of some \( s \in S_{seq} \) and \( o_i^*(c) \) \( \rightarrow^{s(k)} o_i(c) \)
Example

• Mutually exclusive resource; consider sequence
  \( (\text{acquire}_i, \text{release}_i, \text{acquire}_i, \text{acquire}_i, \text{release}_i) \)
  with \( \text{acquire}_i, \text{release}_i \in A_i, (\text{acquire}_i, \text{release}_i) \in R_i; o = \text{acquire}_i, o' = \text{release}_i \)

• \( U_i(1) = (\text{acquire}_i) \Rightarrow n_o(U_i(1)) = 1, n_{o'}(U_i(1)) = 0 \)
• \( U_i(2) = (\text{acquire}_i, \text{release}_i) \Rightarrow n_o(U_i(2)) = 1, n_{o'}(U_i(2)) = 1 \)
• \( U_i(3) = (\text{acquire}_i, \text{release}_i, \text{acquire}_i) \Rightarrow n_o(U_i(3)) = 2, n_{o'}(U_i(3)) = 1 \)
• \( U_i(4) = (\text{acquire}_i, \text{release}_i, \text{acquire}_i, \text{acquire}_i) \Rightarrow n_o(U_i(4)) = 3, n_{o'}(U_i(4)) = 1 \)
• \( U_i(5) = (\text{acquire}_i, \text{release}_i, \text{acquire}_i, \text{acquire}_i, \text{release}_i) \Rightarrow \)
  \( n_o(U_i(5)) = 3, n_{o'}(U_i(5)) = 2 \)

• As \( n_o(U_i(k)) \geq n_{o'}(U_i(k)) \) for \( k = 1, \ldots, 5 \), the sequence is safe
Example (con’t)

• Let $c$ be true whenever resource can be released
  • That is, initially and whenever a $\text{release}_i$ operation is performed

• Consider sequence: $(\text{acquire}_1, \text{acquire}_2^*(c), \text{release}_1, \text{release}_2, \ldots, \text{acquire}_k, \text{acquire}_{k+1}(c), \text{release}_k, \text{release}_{k+1}, \ldots)$

• For all $k \geq 1$, $\text{acquire}_i^*(c) \rightarrow^{s(1)} \text{acquire}_{k+1}(c)$, so this is live sequence
  • Here, $\text{acquire}_{k+1}(c)$ occurs between $\text{release}_k$ and $\text{release}_{k+1}$
Expressing User Agreements

• Use temporal logics

• Symbols
  • \(\square\): henceforth (the predicate is true and will remain true)
  • \(\◇\): eventually (the predicate is either true now, or will become true in the future)
  • \(\sim\): will lead to (if the first part is true, the second part will eventually become true); so \(A \sim B\) is shorthand for \(A \implies \◇ B\)
Example

• Acquiring and releasing mutually exclusive resource type
• User agreement: once a process is blocked on an acquire operation, enough release operations will release enough resources of that type to allow blocked process to proceed

service resource_allocator

User agreement

\[\text{in(acquire)} \sim (\Box \Diamond (\#\text{active\_release} > 0) \lor (\text{free} \geq \text{acquire.n}))\]

• When a process issues an acquire request, at some later time at least 1 release operation occurs, and enough resources will be freed for the requesting process to acquire the needed resources
Finite Waiting Time Policy

• *Fairness policy*: prevents starvation; ensures process using a resource will not block indefinitely if given the opportunity to progress

• *Simultaneity policy*: ensures progress; provides opportunities process needs to use resource

• *User agreement*: see earlier

• If these three hold, no process will wait an indefinite time before accessing and using the resource
Example

• Continuing example ... these and above user agreement ensure no indefinite blocking

sharing policies

fairness

\[(at(acquire) \land \Box \Diamond ((\text{free} \geq \text{acquire.n}) \land (\#\text{active} = 0))) \leadsto after(acquire)\]

\[(at(release) \land \Box \Diamond (\#\text{active} = 0)) \leadsto after(release)\]

simultaneity

\[(in(acquire) \land (\Box \Diamond (\text{free} \geq \text{acquire.n})) \land (\Box \Diamond (\#\text{active} = 0))) \leadsto\]

\[((\text{free} \geq \text{acquire.n}) \land (\#\text{active} = 0))\]

\[(in(release) \land \Box \Diamond (\#\text{active}_{\text{release}} > 0)) \leadsto (\text{free} \geq \text{acquire.n})\]
Service Specification

• Interface operations
• Private operations not available outside service
• Resource constraints
• Concurrency constraints
• Finite waiting time policy
Example:

• Interface operations of the resource allocation/deallocation example

### Interface operations

#### acquire($n$: units)

**exception conditions:** $\text{quota}[id] < \text{own}[id] + n$

**effects:**
- $\text{free}' = \text{free} - n$
- $\text{own}[id]' = \text{own}[id] + n$

#### release($n$: units)

**exception conditions:** $n > \text{own}[id]$

**effects:**
- $\text{free}' = \text{free} + n$
- $\text{own}[id]' = \text{own}[id] - n$
Example (con’t)

Resource constrains of the resource allocation/deallocation example

resource constraints

1. $\Box((\text{free} \geq 0) \land (\text{free} \leq \text{size}))$

2. $(\forall \text{id}) \ [\Box(\text{own}[	ext{id}] \geq 0) \land (\text{own}[	ext{id}] \leq \text{quota}[	ext{id}]) ]$

3. $(\text{free} = N) \Rightarrow ((\text{free} = N) \ \text{UNTIL } (\text{after(\text{acquire})} \lor \text{after(\text{release})})))$

4. $(\forall \text{id}) \ [ (\text{own}[	ext{id}] = M) \Rightarrow ((\text{own}[	ext{id}] = M) \ \text{UNTIL } (\text{after(\text{acquire})} \lor \text{after(\text{release})})))$
Example (con’t)

Concurrency constraints of the resource allocation/deallocation example

**Concurrency constraints**

1. $\square (#\text{active} \leq 1)$
2. $(#\text{active} = 1) \rightsquigarrow (#\text{active} = 1)$
Denial of Service

• Service specification policies, user agreements prevent denial of service *if enforced*

• These do *not* prevent a long wait time; they simply ensure the wait time is finite
Quiz

A process waits for 10 hours to access a resource. Is this a denial of service attack?

• No, as the process got the resource
• It depends on the policy describing the service expected
• Yes, as the fairness constraint was not satisfied because of the long wait