Homework 2 - Solutions

CS 414, Spring 2011
Instructor: Klara Nahrstedt

Note: Homework is an individual effort, i.e., **no working in groups**. Consider the homework as a preparation for your final. The deadline for HW2 is **Monday, May 9, at 11:59pm**. You can submit your solutions through Compass in pdf format or you can slide the homework solutions in paper form under the door of the office 3104 Siebel Center. The homework is 100 points together.

Problem 1: Multimedia CPU Scheduling (25 Points)

1.1 Let *T = (a,b,c)* be the set of threads processing continuous media streams. Assume that the media threads are independent of each other. Let the average processing times *e* of the threads per medium data units be *e(a) = 2 ms, e(b) = 18 ms*, *e(c) = 80 ms*. Let the periods *p* of the media threads be *p(a) = 50 ms*, *p(b) = 200 ms, p(c) = 250 ms.*

(1) **(3 Points)** Are the threads in the set *T* schedulable using **preemptive Rate-Monotonic Scheduling (RMS)** algorithm? Show the schedulability condition and calculation.

Solution: we will use the RMS schedulability test: $\sum_{i=1}^{3} \frac{e_i}{p_i} \leq \ln 2$; (ln 2 = 0.69)

i.e., 2/50 + 18/200 + 80/250 = 40/1000 + 90/1000 + 320/1000 = 450/1000 = 0.45 < ln 2, hence the tasks in set T are schedulable since the schedulability condition is satisfied.

(2) **(5 Points)** If schedule exists from threads in *T*, using **preemptive RMS** algorithm, provide the schedule ordering. If there does not exist schedule, explain why not. Draw the schedule (in case of yes) or show the deadline violation (in case of no).

Solution:

Task ‘a’ has the highest rate, then task ‘b’ and then task ‘c’. Hence task ‘a’ preempts task ‘b’ and ‘c’ when it arrives and task ‘b’ preempts task ‘c’ when it arrives. The hyper-period after which the schedule repeats itself is 1000 ms (which is the common denominator for the terms in the schedulability condition).

The schedule for the full hyper-period is as follows:

Interval 0-50ms = a(2ms), b(18ms), c(30ms);
Interval 50-100ms = a(2ms), c(48 ms);
Interval 100-150ms = a(2ms), c(2ms);
Interval 150-200ms = a(2ms) ;
Interval 200-250ms = a(2ms), b(18ms);
Interval 250-300ms = a(2ms), c(48ms);
Interval 300-350ms = a(2ms), c(32ms);

Interval 350-400ms = a(2ms);
Interval 400-450ms = a(2ms), b(18ms);
Interval 450-500ms = a(2ms);
Interval 500-550ms = a(2ms), c(48ms);
Interval 550-600ms = a(2ms), c(32ms);
Interval 600-650ms = a(2ms), b(18ms);
Interval 650-700ms = a(2ms);
Interval 700-750ms = a(2ms);
Interval 750-800ms = a(2ms), c(48ms);
Interval 800-850ms = a(2ms), b(18ms), c(30ms);
Interval 850-900ms = a(2ms), c(2ms);
Interval 900-950ms = a(2ms);
Interval 950-1000ms = a(2ms);
(schedule repeats itself)

(3) **(2 Points)** Provide the schedulability test for **preemptive EDF** (Earliest Deadline First) scheduling algorithm. Show the work and calculation.

Solution:

We will use the EDF schedulability test: $\sum_{i=1}^{3} \frac{e_i}{p_i} \leq 1$

2/50+18/200+80/250 = 450/100 < 1, hence the threads in T are schedulable via EDF

(4) (**5 Points**) If the threads in *T* are schedulable using **preemptive EDF**, draw a schedule. If the threads in *T* are not schedulable via preemptive EDF, explain why not.

Solution: At each time point 50 ms the deadline of the task 'a' must be compared with other tasks' deadlines waiting to be scheduled. At each time point of 200 ms, the deadline of the task 'b' must be compared with other tasks' deadlines waiting to be scheduled. At each time point of 250 ms, the deadline of the task 'c' must be compared with other tasks' deadlines waiting to be scheduled. Note that the deadlines of tasks 'a', 'b', 'c' are their periods, i.e., *d(a) = p(a), d(b) = p(b), d(c) = p(c)*.
The schedule is as follows – we will outline the schedule for the full hyper-period which is 1000 ms:

Interval	**Task Order/Their Run-Times**	**Comments**
0-50ms	a(2ms), b(18ms), c(30ms)	Tasks ordered according to their deadlines 50ms < 200ms < 250ms
50-100ms	a(2ms), c(48ms)	At this point the schedule must compare the deadline for task 'a' and task 'c' which are in the ready queue, which means: check if d(a) < d(c). d(a) in this interval is 100ms and d(c) is 250 ms, hence task 'a' goes first before task 'c'
100-150ms	a(2m), c(2ms)	d(a)=150 < d(c) = 250ms
150-200ms	a(2ms)	
200-250ms	a(2ms), b(18ms)	250ms=d(a) < 400ms=d(b)
250-300ms	a(2ms), c(48ms)	300ms=d(a) < 500ms=d(c)

300-350ms	a(2ms), c(32ms)	350ms=d(a) < 500ms=d(c)
350-400ms	a(2ms)	
400-450ms	a(2ms), b(18ms)	450ms=d(a) < 600ms=d(b)
450-500ms	a(2ms)	
500-550ms	a(2ms), c(48ms)	550ms=d(a) < 750ms=d(c)
550-600ms	a(2ms), c(32ms)	600ms=d(a) < 750ms=d(c)
600-650ms	a(2ms), b(18ms)	650ms=d(a) < 800ms= d(b)
650-700ms	a(2ms)	
700-750ms	a(2ms)	
750-800ms	a(2ms), c(48ms)	800ms=d(a) < 1000ms=d(c)
800-850ms	a(2ms), b(18ms), c(30ms)	850ms=d(a) < 1000ms=d(b)=d(c) Note: in this case tasks 'b' and 'c' have the same deadline so either could go first after task 'a'. in both orders we have equal number of context switches. I choose task 'b' before task 'c' since 'b' is ahead of task 'c' in the ready queue (task 'b' has been waiting in the ready queue when task 'c' was preempted at time 800 ms, so 'c' is queued after 'b')
850-900ms	a(2ms), c(2ms)	900ms=d(a) < 1000ms=d(c)
900-950ms	a(2ms)	
950-1000ms	a(2ms)	

1.2. **(10 Points)** Assume that message size *M* of the media data unit (frame size of an uncompressed video) of each thread in set *T* is as follows: *M(a) = 100 bytes*, *M(b) = 400 bytes*, *M(c) = 1024 bytes*. Assume the messages arrive for processing to the corresponding buffers (i.e., thread *a* serves stream *a* and buffers data into buffer Buf_a, thread *b* serves stream *b* and buffers data into buffer Buf_b, and thread *c* serves stream *c* and buffers data into buffer Buf_c) in a streaming fashion. How many bytes do you need to receive in order to avoid **starvation** and **overflow** buffer states (specify the number of bytes for each stream *a, b, c*)?

Solution:

To avoid starvation for stream/thread 'a', the amount of received bytes within 50 ms must be $C_a \geqq 100$ **bytes**, it means, we need to receive enough data to process the message of task 'a' within period *p(a)*. The throughput with isochronous transmission mode and constant bit rate must be **Throughput ≧ 100bytes/50ms = 16000bits/second**.

To avoid starvation for stream/thread 'b', the amount of received bytes within 200ms must be $C_b \geqq 400$ **bytes**, i.e., Throughput with isochronous transmission mode/constant bit rate must be **Throughput ≧ 400bytes/200ms = 16000 bits/second**

To avoid starvation for stream/thread 'c', the amount of received bytes within 250 ms must be $C_c \geqq 1024$ **bytes**, i.e., **Throughput ≧ 1024bytes/250ms = 32768 bits/second.**

To avoid overflow, we need the number of received bytes C < frame size M to display plus the capacity of the buffer $C(Buf)$ in bytes.
To avoid overflow for stream/thread 'a', the amount of received bytes $C_a < M(a) + C(Buf_a)$; i.e., $C_a \leq 100 + C(Buf_a)$;
To avoid overflow for 'b', $C_b \leq 400 + C(Buf_b)$;
To avoid overflow for 'c', $C_c \leq 1024 + C(Buf_c)$;

Problem 2: Disk Scheduling (20 Points)

Let us consider a single multimedia disk with a number of tracks from 1 to 100. Let us consider the following set of requests in a request queue for a multimedia disk. Note that each request is represented at the disk management level as a pair of values ***(deadline, track number)***. Let us assume that *request queue* is separate from the *scheduling queue* where the disk scheduling policy operates (makes a decision). Let us assume that the scheduling queue can store 5 items.

The Request Queue and its Order of Requests →
(1, 20), (1, 1), (2, 89), (3, 13), (2, 70), (1, 10), (2, 60), (3, 40), (3, 20), (2, 85),
(3, 35), (2, 76), (3, 96), (3, 36), (3, 95), (4, 48), (4, 1), (4, 50), (3, 5), (3, 45), (4,20).

(1) (10 Points) Specify the scheduling order of these requests when using the **enhanced Scan-EDF**, using differentiating functions $f(N_i)$ when head moves upwards and when head moves downwards. Assume that the disk head starts at the track 20 and starts to move downwards when you start considering the request queue.

Solution: request ordering according to enhanced scan-edf:

To be scheduled requests within each epoch (in queue of 5 items)	**Chosen request from the scheduling queue**	**Decision comments**
(1,20), (1,1), (2,89), (3,13), (2,70)	(1,20)	Head is moving down! We start at track 20, N=20, a) N_i=20, f(N_i) = (20-20)/100, deadline = 1 b) Ni=1, f(Ni) = (20-1)/100, deadline = 1.19 c) Ni=89, then f(Ni)=89/100, deadline = 2.89, d) Ni=13, f(Ni) = (20-13)/100, deadline =3.07 e) Ni=70, f(Ni) = 70/100, deadline, 2.7 So the scheduling order at this point according to perturbed deadlines is (1,20), (1.19, 1), (2.7, 70), (2.89, 89), (3.07, 13) Note: this kind of decision is taken in each step/epoch – I will not show in each step the exact scheduling order, just the decision which request is chosen
(1,1), (2,89), (3,13), (2,70), (1,10)	(1,10)	Head is moving down from N=20
(1,1), (2,89), (3,13), (2,70), (2,60)	(1,1)	Head is moving down from N=10
(2,89), (3,13), (2,70), (2,60), (3,40)	(2,60)	Head turned around, moving upwards, with N=1 and the scheduling order at this point was (2.59, 60), (2.69, 70), (2.88, 89), (3.12, 13), (3.39, 40), so the request (2,60 is chosen)
(2,89), (3,13), (2,70), (3,40), (3,20)	(2,70)	Head moving up, N = 60
(2,89), (3,13), (3,40), (3,20), (2,85)	(2,85)	Head moving up, N=70

(2,89), (3,13), (3,40), (3,20), (3,35)	(2,89)	Head moving up, N = 85
(3,13), (3,40), (3,20), (3,35), (2,76)	(2,76)	Here we have penalty with N = 89, N_i = 76, head moving upwards, then $f(N_i) = N_{max}-N_i/N_{max}$ = (100-76)/100 = 0.24, change of direction was forced to go down!
(3,13), (3,40), (3,20), (3,35), (3,96)	(3,40)	Head moving down, N=76
(3,13), (3,20), (3,35),(3,96), (3,36)	(3,36)	Head moving down, N =40
(3,13), (3,20), (3,35),(3,96), (3,95)	(3,35)	Head moving down, N=36
(3,13), (3,20), (3,96), (3,95), (4,48)	(3,20)	Head moving down, N = 35
(3,13), (3,96), (3,95), (4,48), (4,1)	(3,13)	Head moving down, N=20
(3,96), (3,95), (4,48), (4,1), (4,50)	(3,95)	Here we have again penalty since N = 13, Ni = 95 and head is moving down. So head jumps to 1 and f(Ni) = Ni/100 = 0.95, and starts to move up.
(3,96), (4,48), (4,1), (4,50), (3,5)	(3,96)	Head moving up, N=95
(4,48), (4,1), (4,50), (3,5), (3,45)	(3,45)	Here we have again penalty since N = 96, head moving up, N_i = 45, so $f(N_i) = (N_{max} - N_i)/N_{max}$ = 0.45, head jumps to end of the platter and turns around, moving down!
(4,48), (4,1), (4,50), (3,5), (4,20)	(3,5)	Head moving down, N=45, N_i=5, 1, 20, 48, 50
(4,48),(4,1), (4,50), (4,20),	(4,1)	Head moving down, N = 5, N_i = 1, 20, 48, 50, sorted schedule according to perturbed deadlines: (4.04, 1), (4.2, 20), (4.48, 48), (4,5, 50)
(4,48), (4,50), (4,20)	(4,20)	Head turns around and moves up, N= 1
(4,48), (4,50)	(4,48)	Head moving up, N = 20
(4,50)	(4,50)	Head moving up, N = 48

(2) (10 Points) Specify the scheduling order of the requests above when using the **EDF** disk scheduling policy and compare which disk scheduling policy yields more efficient order of requests in terms of the **number of tracks** that the disk head needs to pass when requests are scheduled with EDF or enhanced SCA-EDF. Assume that the disk head starts at the track 20 and starts to move downwards when you start considering the order of requests in the above queue.

Considered scheduled request in each epoch/step	**Chosen request (sort according to deadline and if requests have the same deadline, pick according to FCFS)**
(1,20), (1,1), (2,89), (3,13), (2,70)	(1,20)
(1,1), (2,89), (3,13), (2,70), (1,10)	(1,1)
(2,89), (3,13), (2,70), (1,10), (2,60)	(1,10)
(2,89), (3,13), (2,70), (2,60), (3,40)	(2,89)
(3,13), (2,70), (2,60), (3,40), (3,20)	(2,70)
(3,13), (2,60), (3,40), (3,20), (2,85)	(2,60)
(3,13), (3,40), (3,20), (2,85), (3,35)	(2,85)

(3,13), (3,40), (3,20), (3,35), (2,76)	(2,76)
(3,13), (3,40), (3,20), (3,35), (3,96)	(3,13)
(3,40), (3,20), (3,35),(3,96), (3,36)	(3,40)
(3,20), (3,35),(3,96), (3,36), (3,95)	(3,20)
(3,35), (3,96), (3,36), (3,95), (4,48)	(3,35)
(3,96), (3,36), (3,95), (4,48), (4,1)	(3,95)
(3,36), (3,95), (4,48), (4,1), (4,50)	(3,36)
(3,95), (4,48), (4,1), (4,50), (3,5)	(3,95)
(4,48), (4,1), (4,50), (3,5), (3,45)	(3,5)
(4,48), (4,1), (4,50), (3,45), (4,20)	(3,45)
(4,48), (4,1), (4,50), (4,20)	(4,48)
(4,1), (4,50), (4,20)	(4,1)
(4,50), (4,20)	(4,50)
(4,20)	(4,20)

Comparison:
Number of tracks for enhanced SCAD-EDF: (20-10) + (10-1) + (60-1) + (70-60) + (85-70) + (89-85) + (100-76) [Note: penalty to consider where head jumps to the end and then starts moving down]+ (76-40) + (40-36) + (36-35) + (35-20) – (20-13) + 95 [Note: penalty to consider where head jumps to the end and starts moving up] + (96-95) + (100-45) [Note: penalty to consider where head jumps to the end and starts moving down] + (45-5) + (5-1) + (20-1) + (48-20) + (50-48) = 438 tracks

Number of tracks for EDF: (20-1) + (10-1) + (89-10) + (89-76) + (76-60) + (85-60) + (85-60) + (85-76) + (76-13) + (40-13) + (40-20) + (35-20) + (96-35) + (96-36) + (95-36) + (95-5) + (45-5) + (48-45) + (48-1) + (50-1) + (50-20) = 714 tracks

So enhanced SCAN-EDF is much more efficient with much less tracks to move through.

Problem 3: Multimedia Networking (25 Points)

Design a transmission system between two desktop clients being located at different coasts of USA, executing multimedia (audio/video) chat service. Assume that you have available all the standard protocols such as RSVP, RTP, RTCP, RTSP, SIP, SDP, UDP, TCP, IP.

(1) **(10 Points)** Propose algorithms, protocols and architectural constructs for the **control plane** for your **chat system**, i.e., explain how you will setup audio/video connections (what functions, protocols, capabilities must be in place to allow for your **chat** service, in what order the function(s)/protocol(s) need to be used, and why did you decide to use particular function(s), protocol(s)) and how you will control/manage connections (what happens in control plane once the video/audio files are being streamed). Differentiate in your description how you deal with audio and video and what needs to be done in the control plane to have the audio and video delivered in synchronized manner.

Solution:

Our design will aim to establish synchronous transmission connection between the chat caller and callee. Note that there are different designs, and other designs will be accepted as long as they are sound!!! (e.g., if you choose to have an asynchronous transmission – best effort over Internet, you need to discuss setup of buffers and management of connection(s), etc. to achieve a chat conversation, so timely conversation must be considered.)

In the control plane we will consider four major parts:

1. **Signaling to find** the chat partners caller/callee - we will use the **SIP protocol** (Session Initiation Protocol) where the caller sends SIP request to the SIP proxy and the SIP proxy on behalf of the caller uses the SIP location service to find callee. This is especially useful if callee is mobile and location service is needed to discover new location of the callee. Once the SIP service finds the callee and callee accepts the chat conversation, the SIP service lets the caller know the IP address of the callee to setup the direct connection between caller and callee for a new chat session.
2. We will enable the **synchronous transmission mode** in the Internet for the chat session, by using the **RSVP protocol**. Using RSVP protocol each sender (caller and callee) for each chat connection (e.g., for audio connection, video connection) sends a PATH request to the receiver with resource/QoS requirements. If PATH with request resources exists between sender/receiver pair, upon the return of the response message from the receiver to the sender, the resources will be reserved for the PATH request. We assume that each chat session will have separate connection for audio and video between sender/receiver pair. Once the RSVP reservations are done, the RTP/UDP/IP transmission over these reserved PATHs can start.
3. We will also establish via **TCP/IP** a direct chat service negotiation protocol between caller and callee to **exchange application metadata parameters** such as Video codec, Audio codec, frame size, frame rate, audio sampling rate, skew parameters, synchronization points (e.g., when to resync audio and video). We will use the SDP (Session Description Protocol) to exchange the needed application metadata needed for the chat application services. We use **SDP over TCP/IP** to ensure reliable exchange of negotiated metadata. This SDP negotiation connection also stays open during the chat if changes need to be made, e.g., change a codec due to disturbances on the network, change in frame rate and frame size.
4. Since our design will use RTP/UDP/IP, we will also utilize **RTCP** to exchange control information for monitoring purposes and possible adaptation at higher layers (although again since we are using RSVP, we will achieve bounds on end-to-end delay (EED) and so bursts can be expected, i.e., early packets can arrive, but the EED upper bound should be preserved.

(1) **(15 Points)** Propose algorithms/functions, protocols, and architectural constructs for the **data plane** for your video/audio **chat** system, i.e., explain how you will stream video between clients, what needs to be considered to provide guaranteed delivery (what functions, protocols, capabilities must be in place to deliver guaranteed chat service to clients, in what order the functions/protocols need to be used, and why did you decide to use the particular functions, protocols, capabilities). Again, make it clear how you deal with audio and video and how you deliver synchronized audio and video.

Solution:

Our data plan design will include the control plane from problem 3.1, i.e., we have RSVP reserved connections between caller and callee for the chat session, each chat has two separate connections for audio and video between sender/receiver pairs. We will rely on RTP/UDP/IP that follows the PATH reserved by RSVP for each connection. The reserved RSVP resources will yield synchronous transmission at the IP level.

1. At the sending end points, we need to deploy **synchronous traffic shapers** such as *token bucket* to regulate the traffic accordingly.
2. At the receiving end-points, we need to deploy **synchronizers** with feedback channels over SDP negotiation connections since we have separate audio and video connections. The synchronizers will check at agreed synchronization points (e.g., every 1 second) if audio is ahead of video or video is ahead of audio or both are in sync within agreed upon skew (e.g., 80ms). The reason we need synchronizers is that some packets can come earlier in bursts and we need to buffer them, some packets can get dropped, etc. If video is ahead of audio, video frames should be dropped and feedback should be sent to the sender via SDP negotiation connection to slow down the video frame rate to an agreed and acceptable rate. If audio is ahead of video, don't drop audio (audio is much more important in chat than video), but send feedback to sender to increase the video frame rate to an agreed and acceptable rate.
3. At the receiving end-points, we will need some minimal **buffering at the application layer** since we can have burst of early arriving packets (synchronous traffic). Note that there should be only smaller buffer size since any buffering contributes to increased end-to-end delay. Depending on the media rate, in interactive applications the receiving buffers should not exceed amount of data corresponding to 100ms (the assumption is that the networking delay is within 10-30 ms and the overall acceptable interactive EED delay to have meaningful conversation is approximately 120-150ms)

Problem 4. (10 Points) Synchronization

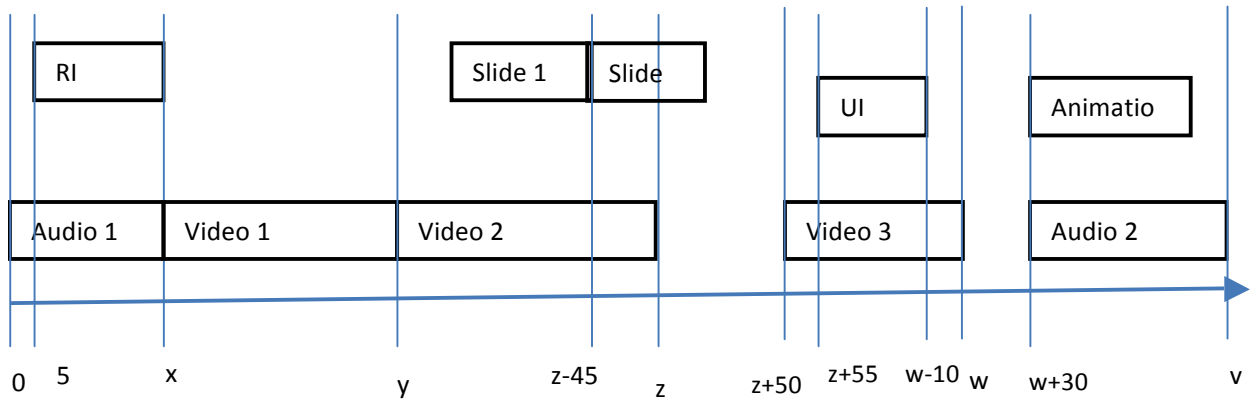
Consider the following Interval-based specification:

Audio1 while(5,0) RecordedInteractions
Audio1 before(0) Video 1;
Video1 before(0) Video2;
Video2 before(50) Video3;
Video3 while (5,10) UserInteraction;
Video3 before(30) Audio2;
Audio2 while(0,0) Animation;
Slide1 before(100) UserInteraction;
Slide1 before(0) Slide2;

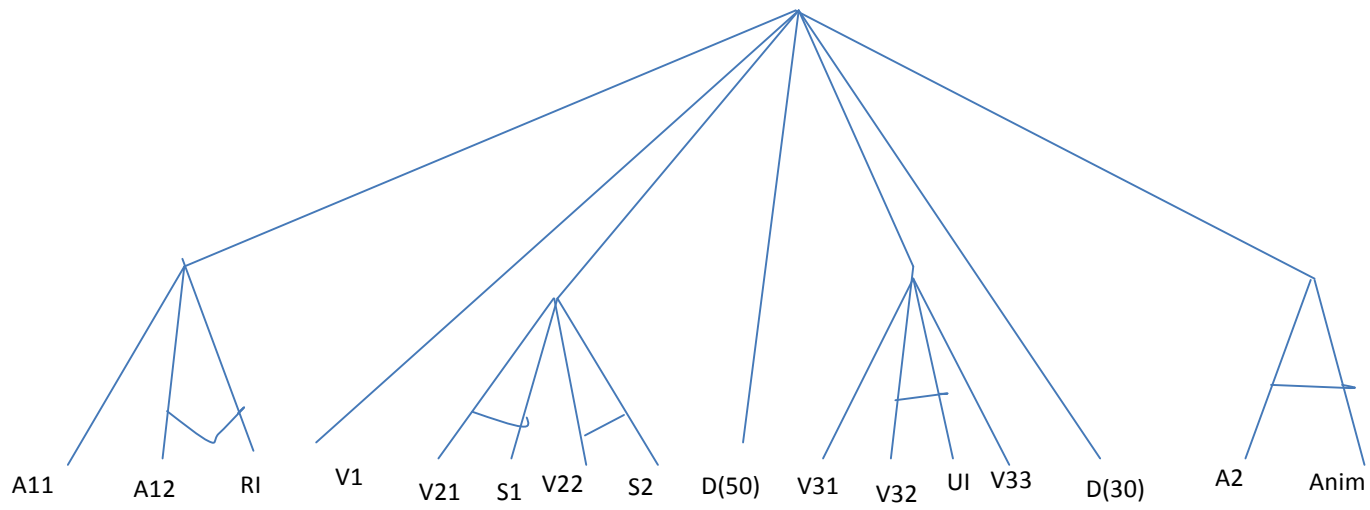
Transform the interval-based specification into (1) time-axis-based specification and (2) hierarchical specification. Argue clearly what synchronization information you could capture with the following (interval-based, time-axis-based and hierarchical) specifications and which synchronization information got lost.

Solution:

Time-axis specification



Since the inter-based specification does not specify duration, this information cannot be regained in the time-axis specification. Hence, we only know that **audio 1**, **RI** respective end at time '*x*', **video 1** ends at time '*y*', **video 2** ends at time 'z', **video 3** starts at '*z+50'* and ends at time 'w', and **audio 2** starts at w+30 and ends at time 'v'. Similar, we do not know when **slide 1** begins and when **slide 2** ends. The only new information we have that **slide 1** ends and **slide 2** begins at 'z-45', and is tied to the end time 'z'. Now, since we do not know how long **video 2** is playing, we cannot express if it slide 1 finishes/slide 2 begins during **video 2** playtime or already during **video 1**. In our drawing, we assume that 'z-45' falls between 'y' and 'z', but it is only an assumption since there is not enough other information to specify the point more precisely. Relations such as **Recorded Interactions (RI)** starts 5 time units later than **audio 1** and finishes at the same time can be well expressed, i.e., **RI** finishes at time 'x', **User Interactions (UI)** start at 'z+55' and finish at 'w-10', again being dependent on the end time 'z' of **video 2,** start time 'z+50' of video 3 and end time of video 3 'w'. **Animation** has the same start time 'w+30' as **audio 2** and the same end time 'v' as **audio 2**.



Hierarchical Specification:

With the hierarchical specification, we are losing even more information, since we cannot express that RI start exactly 5 time units after audio 1 begins. We know that there is a piece of audio 1 (A11) that plays between A12, but we don't know how long. Similar there is no way how to specify and know when slide 1 begins. We assume that slide 1 does not play during V1 and that is plays concurrently with V21. Similar argument and imprecision is introduces with respect to slide 2 (we don't when it ends) and so the specification only captures that there is part of V2 (V22) that plays concurrently with S2. We can express the sequential delay that when V2 finishes, 50 time units pass before the video 3 (V31) starts with D(50). Again, we loose the information how to express that UI starts 5 units after video 3. So the only way how to express some of the relations is to break video 3 into three parts to show that there is piece of video V31 that plays without UI and then part of the video 3 (V32) plays in parallel with UI and part of the video 3 (V33) again plays on its own. Again D(30) shows that between end of video 3 and start of audio 2 (A2) there is 30 times units delay. At the end audio 2 and animations are shown to play concurrently.

Problem 5: Peer-to-Peer Streaming (20 Points)

Consider one VOD server and three peers (P_1, P_2, P_3) that share playback of the same video clip *M1* of the **duration 10 minutes**, i.e., each peer can start to play the same 10-minute video clip *M1* at certain times. Assume that the video stream *M1* is divided into chunks where one chunk is equal to 600 frames. Assume that the video clip *M1* is recorded with the frame rate **20 frames** per second, and is compressed with Motion JPEG (MJPEG). Assume that once one of the peers has some of the *MP1* chunks, the other peers which start later will retrieve the *M1* chunks from their peers (not from the VOD server).

Consider three different ***time cases***: (1) all peers start to play video clip *M1* at ***8 pm***, (2) peer P_1 starts playing *M1* at **time 8 *pm***, peer P_2 requests *M1* 1 **minutes after P_1,** and peer P_3 requests *M1* **3 minutes after P_1**, (3) peer P_3 starts playing *M1* **at time *8 pm***, peer P_1 requests *M1* **4 minutes after P_3**, and P_2 requests *M1* **2 minutes after P_1**.

(a) (**10 Points**) Specify for each ***time case*** (1) how do the peers find each other, (2) how do peers maintain consistent membership list ***(we assume that once the peers register they stay on – no churn)***

Case 1:

Each peer P1, P2, P3 register at the beginning with the server S and the server keeps the membership peer list. In this case since all peers play the video at the same time and with the same frame rate, they download the M1 chunks from the server throughout the playback of the M1 10 minutes video and do not need to know about each other, so we don't need any discovery protocol for the peers to find each other (since there is no churn after nodes register). Once the peers register, the only thing they need to know the address of the server S to download the chunks concurrently.

Case 2.

At 8pm P1 registers with server S, and S starts to create a membership peer list (i.e., {P1}).

At 8:01pm, peer P2 registers with server S, S updates membership list, i.e, {P1, P2}, and informs both peers what the peer list is, i.e., P1 will know about P2 and P2 knows about P1. P2 contacts P1 to get the list of chunks that P1 has from S. After this initial phase, P1 retrieves chunks from S, and P2 retrieves chunks from P1 for streaming. Since there is no churn rate, peer membership list is stable and P2 keeps retrieving chunks from P1 until the end of M1. If P1 finishes playing M1, it stays on until it receives 'end' packet from P2. As peers exist, they inform S and S updates the membership list, removing exiting peers from the list. Since peers wait for each other to finish through the 'end' message, S does not have to send updated membership list, but for completeness it can (it represents an overhead).

At 8:03pm, P3 registers with server S, S updates membership list {P1, P2, P3} and informs all peers about the updated membership list. P3 contacts P1 and P2 to get information what chunks they have when P3 joins. After peers know all the peers who join the session for the movie M1 and who has what chunks, we assume that the peers create dependencies as follows, P2 retrieves chunks from P1 and P3 retrieves chunks from P2, and P1 retrieves chunks from S. (Note that there could be also a different (correct) dependency of peers, e.g., P2 and P3 could retrieve all chunks from P1. However, in this case, P1 would need to keep large buffers.) Once the peer dependency is set, since there is no churn, the membership list is stable and no additional membership update protocol is needed. In terms of the chunks, P1 could let P2 know in-bound (in chunk's data header) what chunks it retrieved from the server, and similarly P2 could let know P3 in-bound what chunks P2 has available for P3 over the time. P2 should

wait (even after finishing playback of M1) for 'end' message from P3 when P3 finishes its playback. Once P2 receives 'end' message, P2 exits as well.

Case 3:

P3 registers with the server at 8am, S keeps membership list {P3}, at 8:04, P1 registers with S and S will update the membership list to {P3, P1}, and S informs all peers about the updated membership list. Then P1 accesses P3 to get information what chunks P3 has at 8:04pm. At 8:06, P2 registers to S, S updates membership list to {P3, P1, P2} and informs all peers about the changed membership list. P2 access P1 and P3 to get information about chunks that P3 and P1 have at time 8:06pm. After this, P2 decides from which peer to retrieve the information. We will choose P2 retrieves from P1.

Since there is no churn, the membership list is stable during the playback. Only once the movie M1 stops playing at P3, then at P1, it is important to finish it cleanly. P3 waits for 'end' message from P1 to exist the system in which case S updates the membership list to {P1, P2} and informs the other peers about the list update. When P1 finishes playback of M1, it waits for the 'end' message from P2 and then exits, informs S, and S updates the membership list. Similar once P2 finishes, it informs S about the exit and S updates the membership list.

(b) (**10 Points**) Specify for each ***time case*** what chunks will be retrieved from whom, at what time and what size of buffers each peer should maintain.

Each chunk is 600 frames which represents 30 seconds of video (at rate 20fps of MJPEG video).

Case 1:

In this case, the size of the buffer could be just a **single 1 chunk buffer** retrieving chunks from the server S since all start at the same time. With single buffer, we assume that the 'retrieval' task works concurrently with the 'display' task dowloading the chunk into the buffer from S and displaying frames in timely fashion, i.e., writing frames into the display buffer

Case 2:

Note: the buffers we are concerned here are upload buffers for chunks on each peer to make sure that we upload correctly needed chunks at the right time. We assume that if chunks are to be played on the peer, the 'display' task retrieves the frames within the chunk time (30 seconds), so that we don't have the situation that peer downloaded the chunk, but the display task did not play it within then same time frame.

At 8:01pm, peer P1 will play 2 chunks (chunk1, chunk2) when peer P2 joins.
At 8:03pm, peer P2 will play 4 chunks (chunk1, 2, 3, 4) when P3 joins

If we consider P2 gets chunks from P1 and P3 gets chunks from P2, then P1 must have **buffers** containing **3 chunks** at one time, and P2 must hold **5 chunks** with the assumption that the chunks will be retrieves in chunk period fashion, i.e., every 30 seconds one chunk from P1 to P2 and one chunk from P2 to P3. P3 has **1 chunk buffer size**.

Case 3:

At 8:04pm, peer P3 will play 8 chunks. At 8:06pm, peer P1 will play 4 chunks and peer P3 12 chunks, when P2 starts.

If we consider that P1 gest chunks from P3 and P2 gets chunks from P1, then P3 must have **9 chunks buffer size**, P1 must have **5 chunks buffer size** to serve P2. P2 can have at least **1 chunk** buffer size.