1. Considering a hybrid threading model, if given a uniprocessor machine for each worker thread, there would be different multiplexing choices. Also considering which thread is I/O bound or CPU bound as an I/O bound thread would most likely use many to 1 multiplexing and a CPU bound thread would most likely use 1 to 1 multiplexing. In a uniprocessor machine for thread 1, I would choose many to 1 multiplexing as it is handling multiple attendee exchanges and puts them into one queue. In a uniprocessor machine, it will handle the exchanges and put them in the queue one at a time. Many to 1 multiplexing in a nutshell is just handling multiple inputs into a combined single output. So in this case we are handling multiple inputs of exchanges into an output queue. For threads 2 and 3, given a uniprocessor machine, I would choose 1 to 1 multiplexing as this thread can be considered CPU bound due to the fact that a large language model will have restrictions depending on the cpu, especially if it has one core. Threads 2 and 3 use a large language model so it will analyze the different exchanges and output into the queue which will take a lot of power from the cpu. Thread 4, given a uniprocessor machine, will use many to 1 multiplexing as it is taking in each summary and outputting to a queue. This thread can be considered I/O bound due to the fact that we are using an AI algorithm that would probably use outside sources to help translate. Because of this, we would most likely use many to 1 as we can combine the inputs and share the channels. For thread 5, I would choose many to 1 as it is sending in multiple translated summaries to a queue to the attendees. This thread can also be considered I/O bound because it will analyze the data based on the language and we will need multiple channels for multiple inputs. Now it would be different if given a multiple CPU core machine because we can do processes concurrently for effectively. For thread 1 I would choose many to 1 multiplexing because it is more efficient as there are multiple cores to help combine multiple inputs. I would also use many to 1 for threads 2 and 3 because as I said for thread 1, we can use multiple cores to efficiently combine the exchanges into a queue. This can also be said for thread 4 as we can process multiple summaries concurrently for translation more efficiently spread out on multiple cores. For thread 5, I would also choose many to 1 multiplexing as we can send out multiple translated summaries to multiple languages concurrently which is more efficient as the attendees can get them. As you can see, given a multiple core CPU, it is much more understandable to use many to 1 multiplexing as we can spread out processes to multiple cores making it much more efficient.

2.

Process A: 5ms (CPU), 7ms (I/O), 6ms (CPU)
RR with 3ms time quantum
P1: 0->3, 3->10, 10->13, 10-> 15, 15->18

SJF = 5 + 7 + 6 = 18
Wait Time
No wait time for first process
Wait time for second is 3ms due to time quantum
Wait time for third is 6 ms
Another time quantum would mean another 6ms wait time so
3 + 6 + 6 = 15ms

SJF = 0 + 5 + 12

Process B: 2ms (CPU), 1ms(I/O), 3ms (CPU), 1ms (I/O), 1ms (CPU)
1st cpu burst is completed
2nd is completed
3rd is completed
2+1+3+1+1 = 8

There will be 2ms wait time right at the end because we are using only 1ms out of the 3 in the time quantum

SJF = 0 + 1 + 1 = 2ms

C: 8ms (CPU), 3ms (I/O), 4ms (CPU)
0->3, 3->6, 6->9,

	Turnaround Time			Wait Time	
	RR	SJF		RR	SJF
A	18ms	18ms		15ms	12ms
B	8ms	8ms		2ms	2ms
C	6ms				
Average					

3.

Sensors	Sampling Frequency(Hz)	CPU time required(ms)
Radar	50	5
High-res infrared	20	20
Low-res infrared	25	10

Overall system overhead is .15
50 -> 1/50 = 20ms
20-> 1/20 = 50ms
25-> 1/25 = 40ms
5/20 + 20/50 + 10/40 = 0.25 + 0.4 + 0.25 + 0.15 = 1.05 is not less than or equal to 1 so it is not schedulable.

We can only reduce radar processing time. If we can reduce it to 30 Hz, we can calculate it as:
30Hz -> 1/40 = 25ms
5/25 = 0.2 + 0.4 + 0.25 + 0.15 = 1 which can make it schedulable.

4. Lower the priority number, the higher priority

Executing:
P3: 6
P7: 4
Waiting for cloud storage access:
P4: 5
P34: 4, first in queue
Waiting for web responses:
P21: 5, first in queue
P6: 7
Waiting to be schedule on the cpu:
P8: 5, first in queue
P10: 6

5. 4kb = 4096 bytes, 32 bit space = 4 bytes
   1. Virtual Address: 0x000067E5
   0000 0000 0000 0000 0110 0111 1110 0101
   Page number = 0 0110 = 0x6
   Offset = 0111 1110 0101 = 0x7E5

   Physical frame: 0x2F
   Physical address: 0x2F7E5

   2. Virtual Address: 0x00015A4B
   Page number = 0000 0000 0000 0001= 0x1
   Offset = 0x5A4B

   Page number 0x1 is not valid so we cannot translate.

   3. Physical memory space: 2GB = 2^30 bytes
   Page size : 64kb = 65,536 bytes = 2^16
   Each page table entry : 4 bytes
   2^30 / 2^16 = 2^14 * 2^2 = 2^16 bytes

   We would need 2^16 total bytes to store for the inverted page table.