A) Shortest time remaining

Time	Proccess		Proccess	Burst Time	Turnaround	Waiting Time
0	P1	P1 arrives	P1	9	21	12
1	P1	P4 and P5 arrive	P2	11	30	19
2	P5	P2 and P3 arrive	P3	7	12	5
3	P5		P4	16	47	31
4	P5		P5	6	6	0
5	P5					
6	P5					
7	P5	Complete				
8	P3					
9	P3					
10	P3					
11	P3					
12	P3					
13	P3					
14	P3	Complete				
15	P1					
16	P1					
17	P1					
18	P1					
19	P1					
20	P1					
21	P1	Complete				
22	P2					
23	P2					
24	P2					
25	P2					
26	P2					
27	P2					
28	P2					
29	P2					
30	P2					
31	P2					
32	P2	Complete				
33	P4					
34	P4					
35	P4					
36	P4					
37	P4					
38	P4					
39	P4					
40	P4					
41	P4					
42	P4					
43	P4					
44	P4					
45	P4					
46	P4					
47	P4					
48	P4	Complete				

A) RR

Time	Proccess	Queue		Proccess	Burst Time	Turnaround	Waiting Time
0	P1	[P1]		P1	9	37	28
1	P1	[P4, P5, P2, P3, P1]		P2	11	42	31
2	P4	[P5, P2, P3, P1, P4]		P3	7	34	27
3	P4	[P2, P3, P1, P4, P5]		P4	16	47	31
4	P5	[P3, P1, P4, P5, P2]		P5	6	24	18
5	P5	[P1, P4, P5, P2, P3]					
6	P2	[P4, P5, P2, P3, P1]					
7	P2						
8	P3						
9	P3						
10	P1						
11	P1						
12	P4						
13	P4						
14	P5						
15	P5						
16	P2						
17	P2						
18	P3						
19	P3						
20	P1						
21	P1						
22	P4						
23	P4						
24	P5						
25	P5	Complete					
26	P2						
27	P2						
28	P3						
29	P3						
30	P1						
31	P1						
32	P4						
33	P4						
34	P2						
35	P2						
36	P3	Complete					
37	P1	Complete					
38	P4						
39	P4						
40	P2						
41	P2						
42	P4						
43	P4						
44	P2	Complete					
45	P4						
46	P4						
47	P4						
48	P4	Complete					

```
illem@willem-QEMU-Virtual-Machine:~/Documents/OS/lab9$ gcc -o lab9_co
sumer lab9_consumer.c -lrt
illem@willem-QEMU-Virtual-Machine:~/Documents/OS/lab9$ ./lab9_produce
 5 2
roducer: Shared buffer address = 0xf7e41dccd000 (logical/virtual addr
ss)
roducer: Address of n = 0xfffffbb7e868
roducer: Generated value 0
roducer: Generated value 2
roducer: Generated value 4
roducer: Generated value 6
roducer: Generated value 8
roducer: Finished producing 5 values
illem@willem-QEMU-Virtual-Machine:~/Documents/OS/lab9$
```

```
willem@willem-QEMU-Virtual-Machine:~/Documents/OS/lab9$ ./lab9_consumer
 5 2
Consumer: Shared buffer address = 0xf38c806cc000 (logical/virtual addre
ss)
Consumer: Address of n = 0xfffffb62e9c0
Consumer: Received value 0
Consumer: Received value 2
Consumer: Received value 4
Consumer: Received value 6
Consumer: Received value 8
Consumer: Finished consuming 5 values
willem@willem-QEMU-Virtual-Machine:~/Documents/OS/lab9$
```

9C)

1) In both processes, print the start address of the shared buffer.

The programs print the shared buffer address using:

printf("Producer/Consumer: Shared buffer address = %p (logical/virtual address)\n", (void *)shm);

0xf7e41dccd000 for the producer, 0xf38c806cc000 for the consumer.

2) Was the address printed Logical (virtual) or physical address?

The address printed is a logical (virtual) address. This is because the programs access memory through the virtual memory system. The `mmap()` function returns a virtual address that the process can use to access the shared memory region. The actual physical address is managed by the operating system's memory management unit (MMU) and is not directly accessible to user-space programs.

3) Print the address of n from your running program and also...
The programs print the address of n using:

printf("Producer/Consumer: Address of n = %p\n", (void *)&n);

0xffffbb7e868 for the producer, 0xffffb62e9c0 for the consumer.

4) Find out where it's stored in the .elf file (executable).

The variable 'n' is a local variable within the main function and is stored on the stack at runtime. To find references to it in the ELF file, I used the readelf tool as suggested in the hints. While the variable itself doesn't have a permanent address in the ELF file (as it's a local variable), I can see in the disassembly that it's accessed at a specific offset from the stack pointer. After compiling with debugging symbols, readelf shows that 'n' is a local variable in the main function's stack frame.

5) Did the addresses match (printed from the running program vs the one in the program's elf file)? Why?

No they didn't match. The address printed from the running program is the runtime virtual address in memory after the program has been loaded and memory has been allocated by the operating system. The address in the ELF file is the address relative to the program's load address as specified in the executable.

The variable `n` is a local variable stored on the stack, while the ELF file would contain information about global or static variables. The actual memory layout is determined at runtime by the loader and memory allocator.

6) What is the virtual address of the entry point in your producer and consumer programs?

command run: readelf -h lab9_producer | grep "Entry point"

Entry point address: 0xbc0

readelf -h lab9_consumer | grep "Entry point"

Entry point address: 0xac0