# Tasks for the Problem Session on RTSD 28.12.2023

[RTS Hardware](https://staff.emu.edu.tr/alexanderchefranov/Documents/CMSE443/CMSE443%20Spring2020/Ch2-Laplante.pdf) (slides 83-89), [RTS Operating Systems](https://staff.emu.edu.tr/alexanderchefranov/Documents/CMSE443/CMSE443%20Spring2020/Laplante-Ch3%20updated1.pptx), and [RTS Operating Systems (cont)](https://staff.emu.edu.tr/alexanderchefranov/Documents/CMSE443/CMSE443%20Fall%202023/Laplante-Ch3-1.ppt) (slides 1-10)

1. What is the basis of Flynn’s classification of computer systems? What is the stream of data? What is the stream of instructions? How the stream of instructions is implemented?
2. What are the features of SISD systems? Examples?
3. What are the features of MISD systems? Examples?
4. What are the features of MISD systems? Examples?
5. What are the features of MIMD systems? Examples?
6. What is the kernel of OS? What are the responsibilities of the OS kernel?
7. How a pseudo-kernel can be implemented in C?
8. What is the state-driven code? How it can be implemented?
9. What is the “dining philosophers” synchronization problem?
10. What are the co-routines (cooperative multitasking)?
11. What are the interrupt-driven systems? What is the reentrant code? What actions are done to handle an interruption signal? What is context switching?
12. What is the preemptive-priority system? Foreground-background system? What are the foreground tasks? What are the background tasks?
13. What is the task control block (TCB)? What information does it contain and what for? What states might a process have? What is the blocked (suspended) state? Why is it introduced? What is the state of a process after its activation?
14. What parameters are used to describe a system of the RTS tasks? What is the release time? What is the phase? What is the period? How phase. Period, and release time are related? What is the absolute and relative deadline?
15. What is the cyclic executive? What is the frame? What is the least common multiple (lcm)? How lcm can be found? What are the three conditions for the frame size selection? Why are the reasons for these conditions?
16. What is the Rate-Monotonic (RM) scheduling? How priorities are assigned to the tasks under RM scheduling?
17. What is the theoretical limit for RM-schedulability? How is it calculated? If the limit is exceeded, does it mean that the tasks are not schedulable?
18. What is the Earliest Deadline First (EDF) scheduling? How priorities are assigned in EDF?
19. What is the theoretical limit for EDF-schedulability? If the limit is exceeded, does it mean that the tasks are not schedulable?
20. What is the use of the buffer in the process communication? What is double buffering?
21. How the ring buffer can be implemented? What two pointers are used with the ring buffer and what for? Where the data is written to the buffer? Where the data is read from the buffer? How modular operations are used to manage the ring buffer? What synchronization problems may arise when using a ring buffer by concurrent processes?
22. What is the mailbox? What operations are defined on the mailbox? What is the property of the mailbox operations necessary for providing mutual exclusion?
23. What data structures can be used to track the use of the resources by the processes? What is the critical resource? What is the critical section? What is the critical region? What is the difference between the critical section and the critical region? What is the mutual exclusion synchronization problem? How semaphores can help to solve the problem of mutual exclusion? How the critical section can be implemented using semaphores?
24. How binary semaphores are defined? What is the semantics of P operation? V operation? What is required from the implementation of P and V operations? What is the property of indivisibility (atomicity)? How P and V operations are also denoted? What are the wait and signal operations?
25. How P and V operations can be implemented using polling loops (active waiting)? Does this solution provide atomicity of the operations? What shall be done by a process finding a closed semaphore? What is the sleeping state of a process? When does a process enter the sleeping (blocked) state? When a process is awakened from sleeping and by whom?
26. How binary semaphores can be implemented using mailbox operations?
27. How mailbox operations can be implemented using binary semaphores? How many binary semaphores are necessary to implement the mailbox operations? What for each of them is used and how are they initialized?
28. How semaphores can be used for signaling of events happening? How can be solved the problem of synchronization of the producers and consumers using semaphores? How the semaphores shall be initialized?
29. What are the counting semaphores? What for they are used? What do they count? What is the meaning of a counting semaphore value?
30. How counting semaphores can be implemented using binary semaphores? How many binary semaphores are necessary to implement a counting semaphore? What is the purpose of each binary semaphore used in the implementation and how is it initialized?
31. How atomic operations can be implemented using the test-and-set special machine instruction? How the test-and-set instruction works? Can it be used in the multiprocessor environment? How atomicity can be provided in the uniprocessor environment using the flag register? What flag is used and what for to provide atomicity in the uniprocessor environment?
32. What the deadlock is? What are the four necessary condition for a deadlock arising? How deadlocks can be prevented? Does the use of semaphores allow avoiding deadlocks?
33. How deadlocks can be recognized? How deadlocks can be resolved?
34. What is the resource numbering method to avoid deadlocks? What is its deficiency?
35. What is the Banker’s algorithm to avoid deadlocks? What are the three assumptions of applicability of the Banker’s algorithm? Can a process request additional resources while running? Can the maximal possible resource request be known a priori? What is expected after a process obtained the maximal possible resource? What is the resource state? What is the safe resource state? In what case the operating system satisfies the processes’ resource requests? Is the Banker’s algorithm applicable in the case of multiple resource types?
36. Priority inversion problem, examples. Priority inheritance protocol
37. Priority ceiling protocol
38. Original ceiling priority protocol
39. Immediate ceiling priority protocol

**Definition of semaphore operations on counting semaphore S(value, queue, max-value, initial-value):**

**P(S): if (--S.value < 0) process is blocked and en-queued into S.queue;**

**V(S): if (S.value < max-value) then S.value++; if (S.value <= 0) then activate one of the processes in S.queue (delete it from S.queue and en-queue into dispatching queue);**

**Task 1.** Implement counting semaphore with the help of binary (binary semaphore is a counting semaphore with max-value=1).

Assume MaxSem is a maximal value of the counting semaphore, CountSem is a binary semaphore initialized by 0 (closed), GuardSem is a binary semaphore initialized by 1 (open), shared by multiple processes. Then

Const

 Struct counting\_sema{

GuardSem:TBinSem=1;//global binary semaphore, initialized by 1, guards counter

CountSem:TBinSem=0;//global binary semaphore to simulate counting semaphore

 R:integer=MaxSem;//global integer, representing current value of counting

//semaphore; negative values give the number of suspended processes

 } Count\_sema;

CP(counting\_sema &count\_sema){//counter P operation

 With (count\_sema){

 P(GuardSem); //lock counter

 R:=R-1; //decrement semaphore value

 If (R<0){//none available?

 V(GuardSem);//release counter

 P(CountSem);//wait for free resource

 }

 V(GuardSem);//release counter

 }

}

CV(counting\_sema &count\_sema){//counter V operation

 With (count\_sema){

 P(GuardSem);//lock counter

 If (R<MaxSem) R:=R+1;//free resource, if possible

 If (R<=0) //any task waiting for free resource

 V(CountSem);//give that task to go ahead

 Else V(GuardSem);//release counter

 }

}

For V operation, releasing of GuardSem in the case of R<=0 is made by activated process (last V(GuardSem) in P operation).

**Task 2.** (Laplante, 2nd Ed., p. 187, Ex. 3 ; 3rd Ed., Ex. 3.21, p. 159). Why it is not wise to disable interruptions before the while loop in the following implementation of P-operation on binary semaphore S:

Procedure P(var S:Boolean);

Begin

 While s=false do;

 S:=false

End;

Procedure V(var s:Boolean);

Begin

 S:=true

End;

If to disable interruptions before *while* loop:

Procedure P(var S:Boolean);

Begin

 Asm

 Cli

 End;

 While s=false do;

 S:=false;

 Asm

 Sti

 end

End; //It is a Pascal code

then in the case of false value of *s* it couldn’t be set to 1 because control yielding will not be allowed, and deadlock could happen. From the other hand, if to disable interrupts after while loop and before assignment, it is not reasonable because control yielding to a next process can happen before disabling of interrupts, and the next process may also see true value of s. So, disable interruptions after while is useless. But without such disabling we can get several processes watching the same not-modified yet value of s. What to do? Solution follows:

Procedure P(var S:Boolean);

Begin

 While true do begin //infinite loop

 While s=false do;//wait for true value of s

 //this loop is made with enabled interrupts!

 Asm //disable interruptions

 Cli

 End;

 If s then begin //again check for true value

 S:=false; //reset if s was true

 Asm

 Sti //enable interruptions

 End;

 Break //exit infinite loop

 End //end of then branch

 Asm

 Sti //enable interruptions

 End; //here we come if s became false in between while loop and interrupt

 //disabling

 End //end of infinite loop

End;//end of P

Simpler solution, but requiring more interruption switches:

Procedure P(var s:boolean);

Var

 Temp:Boolean;

Begin

 Repeat

 Disable interrupts;//cli

 Temp:=s;

 S:=false;

 Enable interrupts;//sti

 Until temp=true;

End;

Procedure V(var s:boolean);//for both cases above of P operation

Begin

 S:=true

End;

**Task 3.** Implement a pipe mechanism of inter-processes communications using circular buffer and semaphores. Consider non-blocking interactions (Similar to Laplante, 2nd Ed., p. 188, Ex. 15)



CircularBuffer has N places for information portions (bytes, sectors, blocks, etc.). We use 2 pointers associated with CircularBuffer: Head, Tail. Tail points to a position where a next portion of information is to be written, Head points to a position from which a next portion of information is to be read. After reading a portion of information, we assume it is not available any more for other requests, and its place may be reused after read operation termination. Circular means that after writing into the 1st position we write into the 2nd and so on, and after writing in the last N-th position we shall try to write in the 1st position; similar for reading. Places for the next write and read operations are pointed by Tail and Head respectively. For circulating, we use (x mod N) operation (remainder after integer division by N).

Simultaneous reading and writing are prohibited, for this sake we use a binary semaphore, BufferSem, initialized by 1 (open).

Pascal code follows:

Const

 N=20; //buffer on 20 places

 Head:integer=1;

 Tail:integer=1;//Head and Tail at first show to the same position, that corresponds to

 //empty buffer state

Type

 Buffer=array[1..N] of Elements;//type of buffer

Var

 BufferSem: BinSemaphore=1;//global semaphore variable shared by several processes

 CircularBuffer:Buffer;

 function WriteBuff( Info:element):integer;//return -1 if no place to write

 begin

 P(BufferSem);

 If Tail mod N+1<>Head then begin //there is space for next write

 CircularBuffer[Tail]:=info; //write into tail position

 Tail:=Tail mod N+1;//shift tail pointer

 result:=0;//successfully wrote

 End

 Else

 result:=-1;//no space in buffer

 V(BufferSem);

End;

function ReadBuff( var Info:element):integer;//return -1 if buffer is empty

 begin

 P(BufferSem);

 If Head<>Tail then begin //buffer is not empty

 Info:=CircularBuffer[Head];//read from head

 Head:= Head mod N+1;//shift head pointer

 result:=0;//successfully read

 End

 Else

 result:=-1;//empty buffer

 V(BufferSem);

End;

Producer uses

If WriteBuff(information)=0 then begin

 Work in the case of successful write operation

End

Else begin

 Work in the case of not writing successfully

End

**Task 4.** Assume that a system has 5 processes and resources of 1 type: processors (total available number is 10). Processes’ resources required and maximal required are as follows:

|  |  |  |
| --- | --- | --- |
| Process | Processors required | Processors maximal required |
| 1 | 1  | 3 |
| 2 | 2  | 2 |
| 3 | 1  | 2 |
| 4 | 2  | 4 |
| 5 | 2  | 3 |

Use Habermann’s algorithm to decide safety of granting required processors

Available processors r=2

B1-c1=2<=r=2 ? yes -> S={P1}

B2-c2=0<=r+c1=3? Yes -> S={p1,p2}

B3-c3=1<=r+c1+c2=2+1+2=5? Yes -> S={p1,p2,p3}

B4-c4=2<=r+c1+c2+c3=5+1=6? Yes -> S={p1,p2,p3,p4}

B5-c5=1<=r+c1+c2+c3+c4=6+2=8? Yes -> S={p1,p2,p3,p4,p5}

Since all the processes are in the sequence, the state is safe, and required resources can be granted