Some exam questions for System Construction

1. *Assume you use the Raspberry Pi as a computer to monitor a flight. You have good basic runtime system such as Minos in place but you don't know about the quality of the code written for the different real time task. In particular you want to be protected against getting stuck in an infinite loop.*

   (a) is this at all a problem in Minos?

   if high-priority task has infinite loop it becomes a problem (other tasks are pre-empted)

   (b) How would you protect yourself?

   Watchdog, Formal Verification

2. *Suppose you had a, say, encryption chip on your system supporting SPI. Assume you had to read a 32-bit secret via SPI (sending a challenge-response). Explain on the → lowest level what happens (draw waveform)*

   assume encryption chip reads secret (like password-authentication)

   CS low, MSB first, MOSI high if bit i = 1, toggle clock, sample on rising edge

3. *What is polarity and phase?*

   polarity: clk idle low/high

   phase: sample on rising edge (0) or falling edge (1)

4. *Properties of the ARM instruction set?*

   32-bit fixed length, 3 operands, conditional execution (almost any instruction has conditional execution bits allows e.g. CMP, SUBGT (if greater subtract)

5. *Explain the module loading concept of Oberon. What happens when a command like A.B gets executed?*

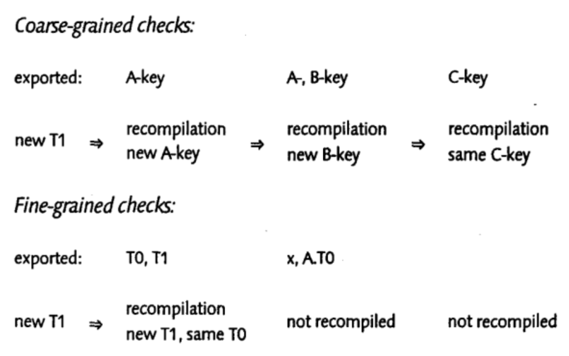
   Modules are loaded on demand, statically linked modules are loaded on start-up. If command gets executed the runtime checks if A is loaded and if not, he initialize all imported modules, executes body of A and then executes A.B

6. *Assume modules A, B and C where C and B both import A. How do you make sure that when A is changed, inconsistent code in B and C is not executed?*

   Create fingerprint of object file, save fingerprint of A in object file of B and C, changes are detected if B or C try to load A again as the fingerprint is different

   What if you want to make this more fine-grained: inconsistencies only in affected portions of the code?

   export fingerprint of exported elements (e.g. types):



Coarse-grained checks:

exported:		A-key		A-, B-key		C-key
new T1	⇒	recompilation new A-key	⇒	recompilation new B-key	⇒	recompilation same C-key

Fine-grained checks:

exported:		T0, T1	x, A.T0	
new T1	⇒	recompilation new T1, same T0	not recompiled	not recompiled

   (see here p. 22)

   What goes to the symbol file and what goes to the object file?

   symbol file -> interface, object file -> code & data

7. *Single processor system (e.g. Minos) Sketch the Classical, simple memory layout?*

   heap from bottom, stack from top

Why unmapped page between top and bottom?

stack overflow protection

What is the difference between stack- and heap- memory in that context?

heap-memory allocation is explicit, while stack-memory is usually allocated implicitly

8. *Multi processor system (e.g. A2). Simple memory layout?*

heap as single shared area directly mapped to physical addresses, each process has own stack on fixed address in virtual memory

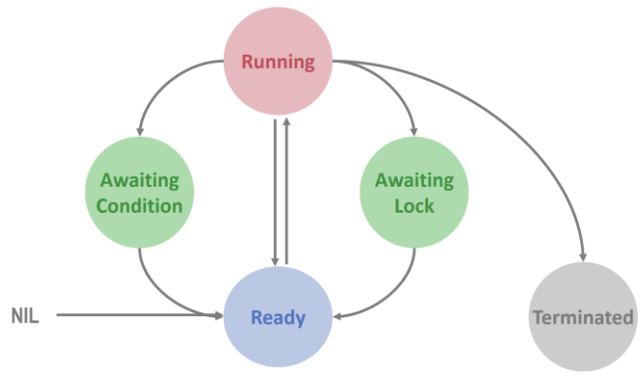
Stacks?

initially only one page mapped to physical memory, if page fault more memory is allocated

How many processes possible?

Limited by memory size as no swapping

9. *Briefly sketch the live cycles of activities in A2.*



Where are the synchronous / asynchronous context switches?

Sync: terminated (running -> terminated), yield (running -> ready), await condition, await lock

Async: timeslicing

10. *Single processor, pre-emptive system, regular tasks with priorities. Single stack?*

Single stack is enough as tasks with higher priority are on top of tasks with lower priority. On each interrupt scheduler checks if task is in correct queue/interval and then schedule it

11. *How can a multiprocessor system be started?*

boot processor runs firmware boot loader, then kernel is loaded, APIC config is read and each processor is started with APIC message, then per processor initialize control register, memory management etc.

12. *The Oberon System presented in the third part of the course (Paul Reeds lectures) provides a message-based system for the GUI. Messages are identified by a dynamic type test system (dynamic module loading!). This requires fast type tests. How can this be implemented?*

types are identified by unique number (type tag, address of type descriptor), for type check we just compare two addresses which is fast

type descriptor is generated by loader and tag is stored in constant area of module, type descriptor contains tags of all base types

13. *Properties of the ARM architecture particularly suitable for operating systems?*

operating modes with shadowed registers

What is a FIQ / Fast Interrupt on ARM?

takes priority over normal IRQ, reduced overhead as more registers are banked

14. *Briefly explain what SPI is. Explain how the communication works.*

SPI is a synchronous, bidirectional, master-multi-slave, serial bus

CS, MOSI, MISO, CLK

Master pulls CS low, then clock starts toggling and data is read into shift register (MSB first)

15. *Explain the Active Cells Compute model. What is hybrid compilation in that context?*

on-chip distributed system consisting of Cells (scope for isolated process) which are connected to a network (with FIFOs)

we have to use different compilation methods for different part of the code (e.g. program logic of cell normal software compilation, cell net description hardware synthesis)

16. *What is GPIO?*

GPIOs are software-controlled processor pins

How can it be used together with buitin-IO drivers on a typical ARM platform?

a GPIO pin can provide an alternative function (e.g. MISO for SPI) which can be enabled in the GPIO Function Selection Register

17. *What does it mean to activate a command in an environment with dynamic (module) loading, what happens when a command is activated?*

check if module is loaded, if not load and init all imported modules, execute module body, run command

What kind of information is necessary during loading and what kind of information for module compilation?

scenario module B import module A

loading: object file of A with fingerprint when A was compiled, object file of B with fingerprint of B and fingerprint of A when B was compiled (we can check if A is still compatible)

compilation: module B needs symbol file and fingerprint of A

How to check consistency at load time?

fine- and coarse-grained fingerprinting

18. *Why does a procedure that shall be used for an interrupt have to be flagged as interrupt procedure?*

Oberon compiler inserts additional assembly to save processor state etc. (special calling convention)

On Arm an additional parameter has to be supplied, why?

different meaning of the return link

19. *What are inter-processor interrupts required for?*

start/stop processors or for managing cache

20. *How do you avoid race conditions (data races) on commonly used memory resource (such as a global variable)? On a single processor system? On a multiprocessor system?*

single processor: turn off interrupts

multi processor: locks

21. *Sketch the programming model of A2. Semantics?*

Normally processes manipulate shared objects

In A2 a process is encapsulated into object: processes communicate over shared active or non-active objects

{EXCLUSIVE} runs under mutual exclusion, AWAIT waits for condition, {ACTIVE} body is executed in own thread

22. *Explain the difference between synchronous and asynchronous context switches. How can we make use of the information?*

synchronous: process gives up control by himself (calling a function) -> most of state saved by calling convention (saving stack i.e. SP, FP is enough)

asynchronous: via interrupt, we don't know state of process -> need to save more state (more expensive)

23. *Semantics of CAS?*

if old value equal to value override with new value and return old value

How do you implement a spinlock using CAS?

binary variable lock, repeat until return value is 0 and try set CAS(lock,0,1)

24. *Single processor system. How to implement mutual exclusion?*

switch off interrupts

25. *Multi processor system. How to implement mutual exclusion (low level).*

Use spin locks implemented with atomic instructions and shared memory

26. *Explain the difference between the execution model of Java and that of Active Oberon on monitors*

Java uses Signal-and-Continue (prefers running thread) -> while-problem (condition changed between signal and scheduling of waiting thread)

Active Oberon uses Signal-and-Exit i.e. evaluates condition when leaving EXCLUSIVE section which prevents while-problem

27. *When a process is de-scheduled by the timer handler, the full state is somewhere on the stack. Is it enough to store SP and FP or does the state have to be stored somewhere?*

No we have to save link register on IRQ stack (shadowed in SVC mode)

28. *Explain the priority scheduling of A2. How do you make sure that a new process with higher priority than a running thread will actually be executed immediately?*

Time slicing and ready queues for different priorities

29. *Do you remember why the first page of our system was unmapped?*

trap NIL pointer

Why is first 1MB unmapped?

by convention (graphics memory)

Pitfall?

large arrays

More advanced questions

30. *What is the ABA problem?*

single memory location was modified by another thread but running thread assumes overall state didn't change (e.g. counter overflow)

How can it be solved?

DCAS, Garbage Collection, Hazard Pointers

How does thread local storage help?

each thread stores his own hazard pointers, other threads check hazard pointers first

31. *Multi processor system, memory layout. How would you implement stack growing facilities in software? (without the MMU)*

save end of last allocated page in process descriptor, if required space exceeds limit -> allocate new space (with MMU use page faults), allocation code inserted by compiler

32. *What is the difference between pre-emptive and cooperative multitasking?*

pre-emptive: process get suspended by external entity

cooperative: process gives up control by himself

What are the advantages and disadvantages of cooperative multitasking?

+ no expensive context switches

+ hardware independent

- requires special compiler

- overhead by scheduler switch code

What is implicit cooperative multitasking and how does it help for lock-free programming?

guarantees that no more than M processes in uncooperative block (M = # of processors) -> # of hazard pointers to check is limited by M

33. *Lock free scheduler: we are using instances of lock-free queues for the ready list. Do you know the problem that can appear when a thread gets scheduled with: get new process from queue, put currently running process to queue, task switch?*

enqueue current process, another processor dequeue process again and run it on same stack

34. *Given a four character 7-segment display hardware on some FPGA board with the following hardware implementation (in order to save bits) [draw display + multiplexer]. You want to output numbers. Using the active cells model, what would you suggest implementing in software / in HDL?*

Multiplexer HDL?

35. *Assume you have a programming language that is claimed to be safe. What kind of functionality has to be supported in OS development that is unsafe?*

get and set values in address, set and get special registers like processor status register, special data type for low-level memory access (BYTE)

36. *What does an object file in a system with dynamic loading consist of?*

dependency information: imports / exports, entries to be fixed-up, code and data

37. *What is a fixup?*

is a pointer to an address in code which have to update if we load the program

38. *How can a dynamic stack be implemented?*

system with virtual memory (MMU): unmapped pages / page fault handler

system without virtual memory: stack allocation by compiler

39. *Describe a mechanism to check if a system is still alive / to check if a certain process does something "often enough".*

watchdog, check register state in timer interrupt

40. *Describe what has to be done to implement an Unload functionality in a system with dynamic module loading.*

per module reference counter/list of imported modules, if ref counter is zero go through module list, decrease ref counter, deallocate memory

41. *Describe a very simple scheduler that supports tasks with (periodic) high-, low- and (non-periodic) background-priority.*

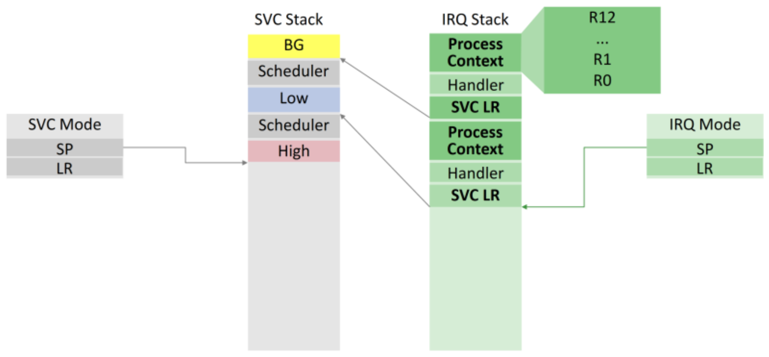
high -> low -> background, timer interrupt calls scheduler, scheduler checks which task is in interval and pre-empt others

What is a sufficient condition to make this scheduler work?

high- and low- tasks have to run to completion within their periods.

Sketch a running scenario and comment on the stack. Is one stack enough?

Yes.



42. *How do you control SPI in software?*

bit-banging, to transfer bit i we shift data i bits to the right, if LSB is 1 then we set MOSI high and toggle the clock

43. *How did the SPI shift registers in the LED display driver allow for multi-digits control?*

different address for each digit

44. *What is a memory-mapped I/O register?*

I/O device is mapped in the same address space as memory

45. *What is an active object?*

object with activity plus a monitor construct

What is the advantage of implementing active objects in the language?

compiler can provide appropriate synchronization and has possibilities for optimization, better cooperation with scheduler

46. *How is a multi-processor system booted?*

One dedicated start processor activating the other processors during system startup

47. What kind of data structures do support threads / processes in a system with a monitor / the active object system?

running array (global), ready queue (global), object queues with conditions and locks (per object), interrupt array (global)

What kind of action takes place during transitions in the process state diagram?

Take lock (global/monitor) before doing any of these!

ready -> running: process is picked from ready queue and put into running queue

running -> ready: enqueue running process into ready queue and schedule new process

running -> cond: enqueue running process into condition queue of object

running -> lock: enqueue running process into waiting queue of object

cond -> ready: if process leaves exclusive section check condition and schedule process

lock -> ready: if process leaves exclusive section schedule waiting process

48. *How can stack allocation be supported in a system with virtual memory for multiple processes?*

each process has his own stack in virtual memory, contiguous pages in virtual, allocate new if page fault

Is the stack conceptually unbounded?

no, virtual memory also has to be divided

49. *In the course we distinguished between synchronous and asynchronous context switches. Why?*

performance for synchronous switches

Explain what has to be done for context switch.

sync->sync: exchange SP, FP, return from Switch

async->sync: set SP, FP, return from Switch

async->async: copy state to stack, return from interrupt

sync->async: copy state to stack, return from interrupt

50. *In A2, certain objects provide a monitor. Where is the monitor information (lock queues etc.) stored?*

in the header of the object, not in the type descriptor

51. *In a system with (active or passive) objects, objects have to provide certain metadata. Which?*

type extension information, method tables, garbage collection flags, lock queues, array size information

Where are the metadata stored?

in type descriptor or in object(header)

52. *What is a type descriptor and what is it used for?*

type descriptor contains all base type tags, address of type descriptor is type tag, type descriptor is generated by loader and is stored in data area of module, used for type checks / type guard

How to implement a type check / type guard?

compare address of expected type descriptor with type tags in type descriptor of type we want to test

53. *When monitors are implemented using constructs such as "Lock", "Unlock" and "Await", behind the scenes fine-granular locks have to be used. Why and which?*

locks on the shared data structures, i.e. lock queues / wait queues

54. *In A2 waiting conditions are evaluated when a process exits the lock of an object. How can that be accomplished?*

wrapping waiting conditions in a helper procedure, condition is re-evaluated with FP

Special question: what can go wrong?

thread-local memory such as the thread id might be wrong during evaluation of the condition

55. *Where are the following metadata typically stored: type tag, array descriptor, lock/ await queues?*

type tag: data section of module

array descriptor: on stack (used to describe arrays with variable length)

lock/await queues: object header

56. *What is the problem with locks?*

Deadlock, Livelock, Starvation, depends heavily on cooperation of competing threads

57. *What is CAS, semantics?*

if old equal a, set a to new value and return old value (executed atomically)

58. *What is the key idea of Cooperative Multitasking wr.t. Lock-free algorithms?*

uncooperative sections, bounded number of processors

What is about interrupts?

modelled as virtual processors

59. What types of parallelism may be exploited using the dataflow approach?

task parallelism, stream parallelism / pipelines and data parallelism (vector operations)

60. *Problems with current commercial multicore processor architecture?*

memory is bottleneck, synchronisation overhead, cache invalidation

61. *What is the advantage of using 18-bits for each instruction on a TRM? Disadvantage?*

high code density but cannot encode all addresses

62. *What is the advantage of dynamically building hardware for dataflow programs?*

less congestion, less energy consumption

63. *If you were to write a hybrid compiler to generate and deploy cell networks on FPGAs, how would you implement it? Where runs what?*

frontend analyses code and backend for each board creates bitstream

frontend creates code for cellnet interpreter (backend), interpreter executes cellnet and create bitstream with hardware library

frontend creates intermediate code, backend creates Oberon executable -> run executable on different backends

64. *What kind of signals do you minimally need in a system of cooperating processes (Active Cells) for point-to-point communication?*

valid, ready, data in / out + select for multiplexing network (ARM AXI4)

65. *What can you say about Cooperative Multitasking / Implicit Cooperative Multitasking?*

cooperative multitasking switches a task only when it's save to do so (save less context), implicit means that the compiler inserts scheduling commands

66. *Explain Object File Format*

header section, imports, commands, entries, datasize, codesize, code, data

67. *How can we achieve mutual exclusion in Minos?*

by disabling interrupts

68. *What data structures are required to be global in order to schedule and run Active Objects?*

ready queue, running queue and interrupt queue

69. How are those global structures implemented (lock-free)?

70. *What are the major differences between HDL and regular programming languages?*

there is a lot more concurrency, timing is explicit, resources are limited

71. *What is ARM Thumb? Why is it implemented? Can Thumb and ARM work together? How?*

16bit instructions with 2 operands, instructions are less powerful, goal of increasing code density (decreased memory cost per functionality), strict subset of ARM IS. Implemented using a dedicated decoder that expands the instructions to the 32bit equivalents (code can be mixed). The CPU can be switched to and from Thumb state.

72. *How do you bring a boot file to the hardware?*

SD card, or via some other (generally writable) memory, possibly internal to the device (e.g. EEPROM)

73. *Describe the advantages of the Active Cells model in comparison to more "conventional" approaches like using General Purpose Architectures or resorting to High Level Synthesis*

modularity and composability, predictable performance (no resource sharing between processors, no time sharing), safety and predictability guarantees (no thread synchronization, no memory sharing, no interrupts), scales massively and predictably, no operating system, implicit data parallelism, low energy consumption. SW-HW codesign, ability to simply use special components ('engines')

74. *Where are the data structures for a scheduler typically stored?*

in kernel's heap

75. *Where are data structures for monitors stored?*

in objects header

76. *Difference between spinlocks and scheduling based locks?*

Assume scheduling-based locks are monitors. Spin locks keeps the thread active (busy waiting). Monitors allow scheduling (thread wait for signal)

77. *Programs shall be written for a target operating system that is not on the host machine. How can this be done without always rebuilding the system / boot image?*

cross-compilation. In a normal system, getting the cross-compiled binary to the target is enough (UART). In Oberon, there are no programs as such and the process is supported by dynamic module loading

78. *How can a system be flashed without direct connection to the hardware via JTAG or the like?*

flash system in pre-boot environment, boot firmware (RPI: running from VC) copies the boot image from any supported storage (RPI: SD) to RAM. In general, the hardware may be such that it looks for the boot image elsewhere (soft boot ROM)