Otázka #9 – File Allocation Table (FAT)

body: 3 z 3

Mějme souborový systém FAT16 s velikostí bloků 512 B. Níže je uvedena relevantní část FAT tabulky.

offset	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9
10	21	58	18	0	-1	-1	43	55	20	0
20	36	44	-1	-1	10	17	0	42	16	34
30	48	-1	27	0	-1	0	28	47	39	45
40	11	22	37	31	25	46	53	-1	32	12
50	0	41	0	-1	-1	23	14	0	30	0

Struktura FAT je již načtená v RAM a přístupy do ní nebudou způsobovat čtení z disku.

Soubor `db.ino` je dlouhý 4856 B a začíná na bloku 40. Aplikace soubor otevře, provede operaci **seek** na pozici 1600 a operací **read** přečte 1536 B. Uveďte indexy všech bloků (ve správném pořadí), které budou touto operací načteny z disku.

Správná odpověď

Odpověď:

30

48

32

27

Question #2 – Generating instructions

points: 3 of 3

Select a fragment of code in C that is the most likely match for the following instructions:

```
ld   $t0,[b]      ; load b
ld   $t1,[c]      ; load c
mul  $t2=$t0,$t1  ; multiplication
ld   $t0,[a]      ; load a
ld   $t1,[d]      ; load d
add  $t2=$t1,$t2  ; addition
mul  $v0=$t0,$t2  ; multiplication
```

Correct answer

- ○ `a * b * c + d`
- ◉ `a * (b * c + d)`
- ○ `a + b + c + d`
- ○ `d * (a << b) + c`
- ○ `a * b * c * d`

Question #1 – CPU ISA

points: 1 of 1

Which items are part of the CPU ISA (Instruction Set Architecture)?

Correct answer

- ☐ thread execution scheduling
- ☐ memory allocation algorithms
- ☐ thread state (context)
- ☐ multitasking
- ☐ function calling convention

Question #7 – Paging

points: 3 of 3

How many page faults may occur at the most when **copying** (i.e., reading and writing) a contiguous `7` MiB block of memory under the following assumptions:

- 32-bit virtual address space, `4` KiB pages, **two-level** page tables, `1024` of `4`-byte items in each paging table
- there are enough free frames, there will be no page replacement
- the first level paging table is always in the memory
- OS implements a very simple algorithm which allocates only one frame during each page failure
- we do not take into account potential page faults caused by instruction fetching (we expect the code is already in the memory)

Correct answer

Answer: 3592

Otázka #3 – Využití asociativní paměti pro cache

body: 1 z 1

Cache pro instrukce a data v CPU je implementována pomocí asociativní paměti, pro kterou platí

Správná odpověď

- ○ klíčem je fyzická adresa, hodnotou je obraz paměti (cache line), vyhledává se v čase O(n)
- ○ fyzická adresa indexuje položku, hodnotou je virtuální adresa, přístup je v konstantním čase
- ○ virtuální adresa indexuje položku
- ○ klíčem je fyzická adresa, hodnotou je obraz paměti (cache line), vyhledává se v čase O(log n)
- ◉ klíčem je fyzická adresa, hodnotou je obraz paměti (cache line), vyhledává se v konstantním čase

Otázka #4 – Zarovnání složených typů

body: 3 z 3

Mějme deklarovaný následující typ struktury **S** a proměnnou **arr** typu pole vytvořené z této struktury:

```
struct S {
  char      input;
  double    precise;
  int       amount;
};
S arr[20];
```

Velikosti základních typů jsou: `double` 8B (64 bitů), `int` 4B (32 bitů) a `char` 1B (8 bitů).

Spočítejte posunutí (offset) **v bajtech** položky `arr[4].amount` od začátku pole `arr`.

Správná odpověď

Odpověď: 112

Otázka #10 – Compare and Swap

body: 0 z 1

Sestavte kód (v jazyce C) těla následující funkce, která reprezentuje sémantiku atomické instrukce Compare-and-swap (CAS), též známé jako Test-and-set lock (TSL).

```
bool CAS(int* var, int oldVal, int newVal) {

    // vaše odpověď

}
```

Pro připomenutí uveďme, že operace vrací `true` při úspěchu, jinak `false`.

Špatná odpověď

Vybrané položky (odpověď)

```
if (*var == oldVal)
return false;
*var = newVal;
return true;
```

Zbývající položky

```
oldVal = *var;
while (*var != oldVal) { /* actively waiting */ }
return *var;
*var = oldVal;
if (*var != oldVal)
```

Otázka #9 – File Allocation Table (FAT)

body: 3 z 3

Mějme souborový systém FAT16 s velikostí bloků `1` KiB. Níže je uvedena relevantní část FAT tabulky.

offset	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9
10	19	47	26	43	0	24	27	41	0	11
20	25	31	0	44	57	42	-1	33	59	-1
30	52	53	48	-1	-1	15	51	40	-1	28
40	39	49	23	36	21	10	0	12	54	55
50	17	30	34	38	13	35	0	29	16	58

Struktura FAT je již načtená v RAM a přístupy do ní nebudou způsobovat čtení z disku.

Soubor `image.png` je dlouhý `8586` B a začíná na bloku `37`. Aplikace soubor otevře, provede operaci **seek** na pozici `4500` a operací **read** přečte `3` KiB. Uveďte indexy všech bloků (ve správném pořadí), které budou touto operací načteny z disku.

Správná odpověď

Odpověď:

59

58

16

27

Otázka #6 – Alokace souvislého bloku paměti

body: 3 z 3

Heap programu (prostor pro dynamickou alokaci paměti) má následující vlastnosti:

- na začátku máme jeden prázdný blok o velikosti `2` MiB na adrese `0xa0000`
- pro alokaci se používá algoritmus *first-fit*, který začíná alokovat od počátku bloku (tj. od `0xa0000`)
- heap používá alokaci po blocích velikosti `32` B

Proces provádí alokaci a dealokaci paměti na heapu v následující sekvenci (alokované úseky paměti jsou označeny pro přehlednost písmeny, která jsou následně použita pro identifikaci uvolňované paměti):

```
A = alloc 1230 B
B = alloc 42 B
C = alloc 314 B
D = alloc 640 B
free A
E = alloc 4 KiB
free D
free C
F = alloc 1234 B
```

Spočítejte, na jaké adrese bude umístěný blok `F`. Pokud zjistíte, že blok `F` nelze alokovat, uveďte jako odpověď číslo `0`.

Správná odpověď

Odpověď:

0xa0000

Otázka #2 – Generování instrukcí

body: 3 z 3

Vyberte fragment kódu jazyka C, který nejlépe odpovídá následujícím instrukcím:

```
    ld   $t0,[b]      ; load b
    ld   $t1,[c]      ; load c
    mul  $t2=$t0,$t1  ; multiplication
    ld   $t0,[a]      ; load a
    ld   $t1,[d]      ; load d
    add  $t2=$t1,$t2  ; addition
    mul  $v0=$t0,$t2  ; multiplication
```

Správná odpověď

- ○ `a * b * c + d`
- ○ `if (a * b) return c + d;`
- ○ `b + c + a * d`
- ◉ `a * (b * c + d)`
- ○ `a * (b * c & d)`

Otázka #8 – Přerušení

body: 0 z 1

Popište průběh zpracování přerušení na CPU.

✗ Špatná odpověď

Vybrané položky (odpověď)

- CPU přeruší aktuální proud instrukcí (mezi instrukcemi) a uloží programový čítač
- na základě zdroje přerušení, CPU určí adresu obsluhy přerušení
- CPU se přepne do systémového režimu a spustí se obsluha přerušení na zjištěné adrese
- CPU obnoví zbytek svého stavu a pomocí speciální instrukce se vrátí k vykonávání původního proudu instrukcí

Zbývající položky

- CPU přečte adresu obsluhy přerušení z datové sběrnice určené pro komunikaci s kontroléry zařízení
- na základě zdroje přerušení, OS určí adresu obsluhy přerušení
- OS uloží stav CPU, obslouží zařízení (nebo cokoli jiného, co způsobilo přerušení), a následně obnoví uložený stav CPU
- CPU se přepne do uživatelského režimu a spustí se obsluha přerušení na zjištěné adrese
- OS přeruší aktuální proud instrukcí (mezi instrukcemi) a uloží programový čítač
- OS přeruší aktuální proud instrukcí (mezi instrukcemi)
- OS obslouží zařízení (nebo cokoli jiného, co způsobilo přerušení)

Otázka #1 – CPU ISA

body: 1 z 1

Co je součástí ISA (Instruction Set Architecture) u CPU?

✓ Správná odpověď

- ☐ souborový systém
- ☐ plánování běhu vláken
- ☑ instrukční sada
- ☐ stav (kontext) vlákna
- ☐ linkování programu

Question #4 – Alignment of compound types

points: 3 of 3

Let us declare the following structure type **S** and a variable **arr** of an array created from this structure:

```
struct S {
  int       level;
  char      spin;
  double    y;
  int       amount;
};
S arr[20];
```

The basic types have the following sizes: `double` 64 bits, `int` 32 bits, and `char` 8 bits.

Compute the offset **in bytes** of the field `arr[9].amount` from the beginning of the `arr` array.

✓ Correct answer

Answer: 232

Question #2 – Generating instructions

points: 3 of 3

Select a fragment of code in C that is the most likely match for the following instructions:

```
ld   $t0,[b]      ; load b
ld   $t1,[c]      ; load c
mul  $t2=$t0,$t1  ; multiplication
ld   $t0,[a]      ; load a
ld   $t1,[d]      ; load d
add  $t2=$t1,$t2  ; addition
mul  $v0=$t0,$t2  ; multiplication
```

Correct answer

- ○ `a * b * c + d`
- ◉ `a * (b * c + d)`
- ○ `a + b + c + d`
- ○ `d * (a << b) + c`
- ○ `a * b * c * d`

Question #1 – Pipeline

Instruction pipeline in the CPU

- ☐ changes the order of instructions
- ☐ increases the latency of instruction execution
- ☐ re-maps the registers
- ☐ increases the number of executed instructions (per unit of time)
- ☐ looks up data in LLC (Last Level Cache)

Question #6 – Allocation of a continuous memory block

points: 3 of 3

Program heap (memory area for dynamic allocation) has the following properties:

- initially, there is one empty block taking `2` MiB located at address `0xa0000`
- the allocation is governed by the *first-fit* algorithm, which starts allocating from the beginning of the block (i.e., from `0xa0000`)
- the heap allocates memory by `32` B blocks

The process performs the following sequence of memory allocation and deallocation operations on the heap (the allocated memory blocks are denoted with letters, which are used when the memory is being released):

```
A = alloc 33 B
B = alloc 1 KiB
C = alloc 8200 B
D = alloc 33 B
E = alloc 33 B
free D
free C
F = alloc 8240 B
```

Compute an address where the block `F` will be located. If the block `F` cannot be allocated, write number `0` as the answer.

Correct answer

Answer: 0xa0440

Otázka #3 – Využití asociativní paměti pro cache

body: 1 z 1

Cache pro instrukce a data v CPU je implementována pomocí asociativní paměti, pro kterou platí

Správná odpověď

- ○ klíčem je fyzická adresa, hodnotou je obraz paměti (cache line), vyhledává se v čase O(n)
- ○ fyzická adresa indexuje položku, hodnotou je virtuální adresa, přístup je v konstantním čase
- ○ virtuální adresa indexuje položku
- ○ klíčem je fyzická adresa, hodnotou je obraz paměti (cache line), vyhledává se v čase O(log n)
- ◉ klíčem je fyzická adresa, hodnotou je obraz paměti (cache line), vyhledává se v konstantním čase

Otázka #4 – Zarovnání složených typů

body: 3 z 3

Mějme deklarovaný následující typ struktury **S** a proměnnou **arr** typu pole vytvořené z této struktury:

```
struct S {
  char      input;
  double    precise;
  int       amount;
};
S arr[20];
```

Velikosti základních typů jsou: `double` 8B (64 bitů), `int` 4B (32 bitů) a `char` 1B (8 bitů).

Spočítejte posunutí (offset) **v bajtech** položky `arr[4].amount` od začátku pole `arr`.

Správná odpověď

Odpověď: 112

Otázka #9 – File Allocation Table (FAT)

body: 3 z 3

Mějme souborový systém FAT16 s velikostí bloků `1` KiB. Níže je uvedena relevantní část FAT tabulky.

offset	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9
10	19	47	26	43	0	24	27	41	0	11
20	25	31	0	44	57	42	-1	33	59	-1
30	52	53	48	-1	-1	15	51	40	-1	28
40	39	49	23	36	21	10	0	12	54	55
50	17	30	34	38	13	35	0	29	16	58

Struktura FAT je již načtená v RAM a přístupy do ní nebudou způsobovat čtení z disku.

Soubor `image.png` je dlouhý `8586` B a začíná na bloku `37`. Aplikace soubor otevře, provede operaci **seek** na pozici `4500` a operací **read** přečte `3` KiB. Uveďte indexy všech bloků (ve správném pořadí), které budou touto operací načteny z disku.

Správná odpověď

Odpověď: 59

58

16

27

Otázka #6 – Alokace souvislého bloku paměti

body: 3 z 3

Heap programu (prostor pro dynamickou alokaci paměti) má následující vlastnosti:

- na začátku máme jeden prázdný blok o velikosti `2` MiB na adrese `0xa0000`
- pro alokaci se používá algoritmus *first-fit*, který začíná alokovat od počátku bloku (tj. od `0xa0000`)
- heap používá alokaci po blocích velikosti `32` B

Proces provádí alokaci a dealokaci paměti na heapu v následující sekvenci (alokované úseky paměti jsou označeny pro přehlednost písmeny, která jsou následně použita pro identifikaci uvolňované paměti):

```
A = alloc 1230 B
B = alloc 42 B
C = alloc 314 B
D = alloc 640 B
free A
E = alloc 4 KiB
free D
free C
F = alloc 1234 B
```

Spočítejte, na jaké adrese bude umístěný blok `F`. Pokud zjistíte, že blok `F` nelze alokovat, uveďte jako odpověď číslo `0`.

Správná odpověď

Odpověď: 0xa0000

Otázka #7 – Stránkování

body: 0 z 3

Ke kolika výpadkům stránek může nejvýše dojít při **přečtení** souvislého bloku paměti o velikosti `17` KiB za následujících předpokladů:

- 32-bitový virtuální adresový prostor, `4` KiB stránky, **dvojúrovňové** stránkovací tabulky, v každé stránkovací tabulce je `1024` `4`-bajtových položek
- máme dostatečné množství volných rámců, nebude docházet k výměně stránek
- stránkovací tabulka první úrovně je vždy v paměti
- OS je používá velmi jednoduchý algoritmus, při každém výpadku stránky je schopen alokovat jen jeden rámec
- neřešíme potenciální výpadky stránek pro načtení instrukcí (předpokládáme, že kód je v paměti)

Otázka #10 – Compare and Swap

body: 0 z 1

Sestavte kód (v jazyce C) těla následující funkce, která reprezentuje sémantiku atomické instrukce Compare-and-swap (CAS), též známé jako Test-and-set lock (TSL).

```
bool CAS(int* var, int oldVal, int newVal) {

    // vaše odpověď

}
```

Pro připomenutí uveďme, že operace vrací `true` při úspěchu, jinak `false`.

Špatná odpověď

Vybrané položky (odpověď)

```
if (*var == oldVal)
return false;
*var = newVal;
return true;
```

Zbývající položky

```
oldVal = *var;
while (*var != oldVal) { /* actively waiting */ }
return *var;
*var = oldVal;
if (*var != oldVal)
```

Otázka #2 – Generování instrukcí

body: 3 z 3

Vyberte fragment kódu jazyka C, který nejlépe odpovídá následujícím instrukcím:

```
ld   $t0,[b]      ; load b
ld   $t1,[c]      ; load c
mul  $t2=$t0,$t1  ; multiplication
ld   $t0,[a]      ; load a
ld   $t1,[d]      ; load d
add  $t2=$t1,$t2  ; addition
mul  $v0=$t0,$t2  ; multiplication
```

Správná odpověď

- ○ `a * b * c + d`
- ○ `if (a * b) return c + d;`
- ○ `b + c + a * d`
- ◉ `a * (b * c + d)`
- ○ `a * (b * c & d)`

Otázka #8 – Přerušení

body: 0 z 1

Popište průběh zpracování přerušení na CPU.

Špatná odpověď

Vybrané položky (odpověď)

- CPU přeruší aktuální proud instrukcí (mezi instrukcemi) a uloží programový čítač
- na základě zdroje přerušení, CPU určí adresu obsluhy přerušení
- CPU se přepne do systémového režimu a spustí se obsluha přerušení na zjištěné adrese
- CPU obnoví zbytek svého stavu a pomocí speciální instrukce se vrátí k vykonávání původního proudu instrukcí

Zbývající položky

- CPU přečte adresu obsluhy přerušení z datové sběrnice určené pro komunikaci s kontroléry zařízení
- na základě zdroje přerušení, OS určí adresu obsluhy přerušení
- OS uloží stav CPU, obslouží zařízení (nebo cokoli jiného, co způsobilo přerušení), a následně obnoví uložený stav CPU
- CPU se přepne do uživatelského režimu a spustí se obsluha přerušení na zjištěné adrese
- OS přeruší aktuální proud instrukcí (mezi instrukcemi) a uloží programový čítač
- OS přeruší aktuální proud instrukcí (mezi instrukcemi)
- OS obslouží zařízení (nebo cokoli jiného, co způsobilo přerušení)

Otázka #1 – CPU ISA

body: 1 z 1

Co je součástí ISA (Instruction Set Architecture) u CPU?

Správná odpověď

- ☐ souborový systém
- ☐ plánování běhu vláken
- ☑ instrukční sada
- ☐ stav (kontext) vlákna
- ☐ linkování programu

Otázka #10 – Semafor

body: 0 z 1

Sestavte kód (v jazyce C/C++) těla následující metody, která reprezentuje sémantiku operace `up` na *Semaforu*. Semafor si drží členské proměnné `counter` (typu `unsigned int` -- tedy celé číslo, které nemůže být záporné) a `queue` (fronta vláken) a identifikátor `thisThread` odkazuje na globální objekt reprezentující aktuálně běžící vlákno (ve kterém je kód prováděn).

```
void up() {

    // vaše odpověď

}
```

✗ Špatná odpověď

Vybrané položky (odpověď)

```
if (counter == 0 || !queue.isEmpty())
{
```

```
thisThread = queue.pop();
```

```
thisThread.block();
```

```
} else {
```

```
counter += 1;
```

```
}
```

Zbývající položky

```
auto thread = queue.pop();
```

```
thread.resume();
```

```
if (counter == 0 && !queue.isEmpty())
{
```

Otázka #8 – Přerušení

body: 1 z 1

Jaké jsou možné příčiny přerušení?

✓ Správná odpověď

- [x] externí (změna stavu pinu procesoru)
- [ ] skok na podprogram
- [x] software (provedení speciální instrukce)
- [ ] linkování programu
- [ ] předání parametrů při volání funkce

Otázka #9 – File Allocation Table (FAT)

body: 3 z 3

Mějme souborový systém FAT16 s velikostí bloků 4 KiB. Níže je uvedena relevantní část FAT tabulky.

offset	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9
10	20	26	0	-1	0	25	-1	-1	30	50
20	22	-1	-1	49	18	17	51	38	-1	42
30	13	0	19	52	47	57	35	21	11	36
40	16	37	33	-1	46	-1	41	48	24	15
50	45	32	59	0	29	40	0	55	0	28

Struktura FAT je již načtená v RAM a přístupy do ní nebudou způsobovat čtení z disku.

Soubor `image.png` je dlouhý 26384 B a začíná na bloku 39. Aplikace soubor otevře, provede operaci **seek** na pozici 10000 a operací **read** přečte 16 KiB. Uveďte indexy všech bloků (ve správném pořadí), které budou touto operací načteny z disku.

Správná odpověď

Odpověď:

35

57

55

40

16

Heap programu (prostor pro dynamickou alokaci paměti) má následující vlastnosti:

- na začátku máme jeden prázdný blok o velikosti `1` MiB na adrese `0x80000`
- pro alokaci se používá algoritmus *first-fit*, který začíná alokovat od počátku bloku (tj. od `0x80000`)
- heap používá alokaci po blocích velikosti `16` B

Proces provádí alokaci a dealokaci paměti na heapu v následující sekvenci (alokované úseky paměti jsou označeny pro přehlednost písmeny, která jsou následně použita pro identifikaci uvolňované paměti):

```
A = alloc 1230 B
B = alloc 42 B
C = alloc 314 B
D = alloc 640 B
free A
E = alloc 4 KiB
free D
free C
F = alloc 1234 B
```

Spočítejte, na jaké adrese bude umístěný blok `F`. Pokud zjistíte, že blok `F` nelze alokovat, uveďte jako odpověď číslo `0`.

✓ Správná odpověď

Odpověď: 0x818c0

Otázka #5 – Linker (spojovač)

body: 1 z 1

Linker má za úkol při vytváření spustitelného programu

✓ Správná odpověď

- ○ přeložit všechny moduly projektu a všechny knihovny
- ○ přeložit všechny moduly projektu a připojit všechny knihovny
- ◉ pospojovat všechny přeložené a potřebné moduly projektu a všechny potřebné knihovny, dopočítat relokace
- ○ pospojovat všechny přeložené moduly projektu
- ○ zavést spustitelný program do paměti

Otázka #4 – Zarovnání složených typů

body: 3 z 3

Mějme deklarovaný následující typ struktury **S** a proměnnou **arr** typu pole vytvořené z této struktury:

```
struct S {
  double    measured;
  char      spin;
  int       number;
};
S arr[20];
```

Velikosti základních typů jsou: `double` 8B (64 bitů), `int` 4B (32 bitů) a `char` 1B (8 bitů).

Spočítejte posunutí (offset) **v bajtech** položky `arr[9].spin` od začátku pole `arr`.

Správná odpověď

Odpověď: 152

Otázka #3 – Hierarchie cache

body: 0 z 1

V rámci vícejádrového CPU platí pro hierarchii cache následující tvrzení:

Špatná odpověď

- [ ] nižší úrovně (číselně) jsou rychlejší než vyšší úrovně
- [ ] nemohou existovat privátní úrovně cache (pro každé jádro)
- [x] lze oddělit cache pro instrukce a pro data
- [ ] L1 cache je stejně rychlá jako registry
- [x] nižší úrovně (číselně) jsou větší než vyšší úrovně

Otázka #2 – Generování instrukcí

body: 3 z 3

Vyberte fragment kódu jazyka C, který nejlépe odpovídá následujícím instrukcím:

```
ld   $t0,[b]      ; load b
ld   $t1,[c]      ; load c
mul  $t2=$t0,$t1  ; multiplication
ld   $t0,[a]      ; load a
ld   $t1,[d]      ; load d
add  $t2=$t1,$t2  ; addition
mul  $v0=$t0,$t2  ; multiplication
```

Správná odpověď

- ○ `a * (b * c & d)`
- ○ `d * (a << b) + c`
- ○ `a + (b << c) * d`
- ○ `if (a * b) return c + d;`
- ◉ `a * (b * c + d)`

Otázka #2 – Generování instrukcí

Vyberte fragment kódu jazyka C, který nejlépe odpovídá následujícím instrukcím:

```
ld   $t0,[b]      ; load b
ld   $t1,[c]      ; load c
mul  $t2=$t0,$t1  ; multiplication
ld   $t0,[a]      ; load a
ld   $t1,[d]      ; load d
add  $t2=$t1,$t2  ; addition
mul  $v0=$t0,$t2  ; multiplication
```

Správná odpověď

- (•) `a * (b * c + d)`
- ( ) `a + (b << c) * d`
- ( ) `a * (b * c & d)`
- ( ) `a * b * c * d`
- ( ) `a + b + c + d`

Otázka #4 – Zarovnání složených typů

Mějme deklarovaný následující typ struktury **S** a proměnnou **arr** typu pole vytvořené z této struktury:

```
struct S {
  double    value;
  int       index;
  char      spin;
};
S arr[20];
```

Velikosti základních typů jsou: `double` 8B (64 bitů), `int` 4B (32 bitů) a `char` 1B (8 bitů).

Spočítejte posunutí (offset) **v bajtech** položky `arr[1].spin` od začátku pole `arr`.

Správná odpověď

Odpověď: 28

Otázka #5 – Překladač

Překladač (programovacích jazyků) přeloží

Správná odpověď

- (•) všechny korektní zdrojové kódy na platné a správně použité instrukce cílového CPU, při špatném vstupu ohlásí všechny zjištěné chyby
- ( ) všechny zdrojové kódy na platné a správně použité instrukce cílového CPU
- ( ) všechny zdrojové kódy na instrukce cílového CPU a pospojuje je do spustitelného programu
- ( ) všechny zdrojové kódy a vyřeší jejich relokace ve všech modulech projektu
- ( ) všechny zdrojové kódy na instrukce cílového CPU

Otázka #6 – Alokace souvislého bloku paměti

body: 3 z 3

Heap programu (prostor pro dynamickou alokaci paměti) má následující vlastnosti:

- na začátku máme jeden prázdný blok o velikosti `2` MiB na adrese `0xa0000`
- pro alokaci se používá algoritmus *first-fit*, který začíná alokovat od počátku bloku (tj. od `0xa0000`)
- heap používá alokaci po blocích velikosti `32` B

Proces provádí alokaci a dealokaci paměti na heapu v následující sekvenci (alokované úseky paměti jsou označeny pro přehlednost písmeny, která jsou následně použita pro identifikaci uvolňované paměti):

```
A = alloc 1230 B
B = alloc 42 B
C = alloc 314 B
D = alloc 640 B
free A
E = alloc 4 KiB
free D
free C
F = alloc 1234 B
```

Spočítejte, na jaké adrese bude umístěný blok `F`. Pokud zjistíte, že blok `F` nelze alokovat, uveďte jako odpověď číslo `0`.

Správná odpověď

Odpověď: `0xa0000`

Otázka #8 – Přerušení

body: 0 z 1

Popište průběh zpracování přerušení na CPU.

Špatná odpověď

Vybrané položky (odpověď)

- na základě zdroje přerušení, OS určí adresu obsluhy přerušení
- CPU přeruší aktuální proud instrukcí (mezi instrukcemi) a uloží programový čítač
- CPU se přepne do systémového režimu a spustí se obsluha přerušení na zjištěné adrese
- OS obnoví stav CPU a pomocí speciální instrukce se vrátí k vykonávání původního proudu instrukcí

Zbývající položky

- na základě zdroje přerušení, CPU určí adresu obsluhy přerušení
- OS přeruší aktuální proud instrukcí (mezi instrukcemi) a uloží programový čítač
- CPU uloží svůj stav, obslouží zařízení (nebo cokoli jiného, co způsobilo přerušení), a následně svůj stav obnoví
- OS uloží stav CPU, obslouží zařízení (nebo cokoli jiného, co způsobilo přerušení), a následně obnoví uložený stav CPU
- CPU obnoví zbytek svého stavu a pomocí speciální instrukce se vrátí k vykonávání původního proudu instrukcí
- kontrolér zařízení přeruší aktuální proud instrukcí CPU (mezi instrukcemi)

Otázka #9 – File Allocation Table (FAT)

body: 3 z 3

Mějme souborový systém FAT16 s velikostí bloků 1 KiB. Níže je uvedena relevantní část FAT tabulky.

offset	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9
10	30	15	-1	48	11	37	54	58	56	0
20	27	26	28	49	0	23	12	25	-1	16
30	-1	51	0	0	0	36	55	59	39	21
40	50	0	20	31	0	35	22	0	1	1
50	52	29	17	42	46	57	54	40	1	14

Struktura FAT je již načtená v RAM a přístupy do ní nebudou způsobovat čtení z disku.

Soubor `file.dat` je dlouhý `9572` B a začíná na bloku `45`. Aplikace soubor otevře, provede operaci **seek** na pozici `6500` a operací **read** přečte `3` KiB. Uveďte indexy všech bloků (ve správném pořadí), které budou touto operací načteny z disku.

Správná odpověď

Odpověď:

50

52

17

58

Otázka #10 – Semafor

body: 0 z 1

Sestavte kód (v jazyce C/C++) těla následující metody, která reprezentuje sémantiku operace `up` na *Semaforu*. Semafor si drží členské proměnné `counter` (typu `unsigned int` -- tedy celé číslo, které nemůže být záporné) a `queue` (fronta vláken) a identifikátor `thisThread` odkazuje na globální objekt reprezentující aktuálně běžící vlákno (ve kterém je kód prováděn).

```
void up() {

    // vaše odpověď

}
```

Špatná odpověď

Vybrané položky (odpověď)

```
counter += 1;
if (counter == 0 && !queue.isEmpty()) {
auto thread = queue.pop();
thread.resume();
}
```

Zbývající položky

```
thisThread = queue.pop();
thisThread.block();
if (counter == 0) {
} else {
thread.block();
```

Otázka #2 – Generování instrukcí

body: 3 z 3

Vyberte fragment kódu jazyka C, který nejlépe odpovídá následujícím instrukcím:

```
    ld   $t0,[a]      ; load a
    ld   $t1,[b]      ; load b
    slt  $t2=$t0,$t1  ; set on less than ($t0 < $t1)
    beq  $t2,0,lbl    ; branch on equal ($t2 == 0)
    st   $t0,[b]      ; store to b
lbl:                  ; label
```

Správná odpověď

- `if (a < b) a = b;`
- `while (b < a) a = b;`
- `if (a > b) a = b;`
- `if (a < b) b = a;` ✓
- `if (a < b) return b;`

Otázka #3 – Využití asociativní paměti pro cache

body: 1 z 1

Cache pro instrukce a data v CPU je implementována pomocí asociativní paměti, pro kterou platí

Správná odpověď

- klíčem je virtuální adresa, hodnotou je obraz paměti (cache line), vyhledává se v čase O(log n)
- klíčem je fyzická adresa, hodnotou je obraz paměti (cache line), vyhledává se v konstantním čase ✓
- klíčem je virtuální adresa, hodnotou je obraz paměti (cache line), vyhledává se v konstantním čase
- virtuální adresa indexuje položku, hodnotou je fyzická adresa, přístup je v konstantním čase
- fyzická adresa indexuje položku

Otázka #4 – Zarovnání složených typů

body: 3 z 3

Mějme deklarovaný následující typ struktury **S** a proměnnou **arr** typu pole vytvořené z této struktury:

```
struct S {
  char      keyPressed;
  double    value;
  int       index;
  int       level;
};
S arr[20];
```

Velikosti základních typů jsou: `double` 8B (64 bitů), `int` 4B (32 bitů) a `char` 1B (8 bitů).

Spočítejte posunutí (offset) **v bajtech** položky `arr[1].value` od začátku pole `arr`.

Správná odpověď

Odpověď: 32

Otázka #5 – Překladač

body: 1 z 1

Překladač (programovacích jazyků) přeloží

Správná odpověď

- všechny korektní zdrojové kódy na platné a správně použité instrukce cílového CPU a pospojuje je do spustitelného programu
- všechny korektní zdrojové kódy na platné a správně použité instrukce cílového CPU
- všechny korektní zdrojové kódy na spustitelný program, při špatném vstupu ohlásí všechny zjištěné chyby
- všechny korektní zdrojové kódy na platné a správně použité instrukce cílového CPU, při špatném vstupu ohlásí všechny zjištěné chyby ✓
- všechny zdrojové kódy na instrukce cílového CPU

Otázka #6 – Alokace souvislého bloku paměti

body: 3 z 3

Heap programu (prostor pro dynamickou alokaci paměti) má následující vlastnosti:

- na začátku máme jeden prázdný blok o velikosti `1` MiB na adrese `0x80000`
- pro alokaci se používá algoritmus *first-fit*, který začíná alokovat od počátku bloku (tj. od `0x80000`)
- heap používá alokaci po blocích velikosti `16` B

Proces provádí alokaci a dealokaci paměti na heapu v následující sekvenci (alokované úseky paměti jsou označeny pro přehlednost písmeny, která jsou následně použita pro identifikaci uvolňované paměti):

```
A = alloc 33 B
B = alloc 1 KiB
C = alloc 8200 B
D = alloc 33 B
E = alloc 33 B
free D
free C
F = alloc 8240 B
```

Spočítejte, na jaké adrese bude umístěný blok `F`. Pokud zjistíte, že blok `F` nelze alokovat, uveďte jako odpověď číslo `0`.

Správná odpověď

Odpověď: 0x80430

Otázka #7 – Stránkování

body: 3 z 3

Ke kolika výpadkům stránek může nejvýše dojít při **kopírování** (tj. čtení a zapisování) souvislého bloku paměti o velikosti `31` KiB za následujících předpokladů:

- 32-bitový virtuální adresový prostor, `2` KiB stránky, **trojúrovňové** stránkovací tabulky
- první stránkovací úroveň má jen `32` 8-bajtové položky, druhá a třetí úroveň mají v každé stránkovací tabulce `256` 8-bajtových položek
- máme dostatečné množství volných rámců, nebude docházet k výměně stránek
- stránkovací tabulka první úrovně je vždy v paměti
- OS je používá velmi jednoduchý algoritmus, při každém výpadku stránky je schopen alokovat jen jeden rámec
- neřešíme potenciální výpadky stránek pro načtení instrukcí (předpokládáme, že kód je v paměti)

Správná odpověď

Odpověď: 42

Otázka #9 – File Allocation Table (FAT)

body: 3 z 3

Mějme souborový systém FAT16 s velikostí bloků `1` KiB. Níže je uvedena relevantní část FAT tabulky.

offset	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9
10	0	0	47	54	30	-1	-1	18	49	50
20	35	42	-1	33	38	12	24	51	59	45
30	44	17	58	-1	14	13	40	26	-1	0
40	48	-1	-1	31	28	16	36	21	56	23
50	-1	-1	27	52	29	46	41	32	15	53

Struktura FAT je již načtená v RAM a přístupy do ní nebudou způsobovat čtení z disku.

Soubor `db.isam` je dlouhý `7000` B a začíná na bloku `20`. Aplikace soubor otevře, provede operaci **seek** na pozici `6500` a operací **read** přečte `500` B. Uveďte indexy všech bloků (ve správném pořadí), které budou touto operací načteny z disku.

Správná odpověď

Odpověď: 16

Otázka #10 – Compare and Swap

body: 1 z 1

Sestavte kód (v jazyce C) těla následující funkce, která reprezentuje sémantiku atomické instrukce Compare-and-swap (CAS), též známé jako Test-and-set lock (TSL).

```
bool CAS(int* var, int oldVal, int newVal) {

    // vaše odpověď

}
```

Pro připomenutí uveďme, že operace vrací `true` při úspěchu, jinak `false`.

Správná odpověď

Vybrané položky (odpověď)

```
if (*var != oldVal)
```

```
return false;
```

```
*var = newVal;
```

```
return true;
```

Zbývající položky

```
while (*var != oldVal) { /* actively waiting */ }
```

```
*var = 0;
```

```
if (*var == oldVal)
```

```
if (*var != newVal)
```

```
return *var;
```