Odpovědi pište na zvláštní odpovědní list s vaším jménem a fotografií. Pokud budete odevzdávat více než jeden list s řešením, tak se na 2. a další listy nezapomeňte podepsat. Do zápatí všech listů vždy napište i/N (kde i je číslo listu, N je celkový počet odevzdaných listů).

Otázka č. 1

Vysvětlete na vhodném časovém diagramu koncept tzv. *diferenciálního přenosu*. Jaké má výhody a nevýhody? Jak by mohl vypadat přenos hodnoty `0xF5`?

Společná část pro otázky označené X

Předpokládejte, že programujeme nějaký jednoduchý operační systém s preemptivním přepínáním vláken pro jednoprocesorovou architekturu x86 (32-bitové CPU s vypnutým stránkováním, tj. logické adresy se přímo rovnají fyzickým adresám), do jehož jádra jsme naimplementovali následující rozšířenou variantu syscallu `CreateThread` (který má navíc argument `arg`, který vloží na připravený zásobník nového vlákna tak, že hlavní procedura vlákna „přijme“ tuto hodnotu jako svůj vlastní argument):

```
type PLongword = ^longword;
     PThreadProc = procedure (i : longint); cdecl;
     TThread = record
       SP : longword; OwnerPID : longword;
       State : longword;
     end;
var currTID, currPID : longword;
    threads : array[0..255] of TThread;

function AllocNewStackTop : pointer; forward;
procedure ExitThread; cdecl; forward;

function CreateThread(
  thrProc : PThreadProc; arg : longint) : longword;
var newTID : longword;
begin
  for newTID := 0 to 255 do begin
    if threads[newTID].State = UNUSED then break;
  end;
  threads[newTID].State := CREATED;
  threads[newTID].OwnerPID := currPID;
  threads[newTID].SP := longword(AllocNewStackTop);
  Dec(threads[newTID].SP, sizeof(longword));
  PLongword(threads[newTID].SP)^ := arg;
  Dec(threads[newTID].SP, sizeof(longword));
  PLongword(threads[newTID].SP)^ :=
    longword(@ExitThread);
  Dec(threads[newTID].SP, sizeof(longword));
  PLongword(threads[newTID].SP)^ := longword(thrProc);
  threads[newTID].State := READY_TO_RUN;
  CreateThread := newTID;
end;
```

Otázka č. 2 (X)

Je výše uvedený kód přenositelný na úrovni zdrojového kódu (source code portable) na libovolnou jinou procesorovou architekturu, pro kterou máme překladač Pascalu? Detailně vysvětlete proč.

Otázka č. 3 (X)

Víme, že funkci `CreateThread` vždy voláme ve stavu s povolenými přerušeními. Pokud je zbytek našeho OS s preemptivním multithreadingem implementován standardním způsobem, je uvedená funkce tzv. *thread-safe*, tj. vysvětlete, zda je pravda, že při jejím běhu nemůže dojít k žádné *race condition*? Pokud race condition může vzniknout, tak navrhněte, jak možnosti jejího vzniku v uvedené funkci zabránit.

Otázka č. 4 (X)

Pokud víme, že níže uvedený příkaz:

```
WriteLn(IntToHex(longword(@threads[0]), 8));
```

Vypíše hodnotu `00419000`, tak popište a vysvětlete, co bude při použití nějakého typického překladače uložené na fyzických adresách `$00419000` až `$0041900F` (jde nám o druh a „identitu“ dat vzhledem k ve společné části uvedeným Pascalovým deklaracím a nikoliv o konkrétní hodnoty).

Otázka č. 5 (X)

Předpokládejte, že jsme uvedený OS upravili pro podporu víceprocesorových systémů, a že pro něj píšeme níže uvedenou aplikaci s procedurou `Process`, která má projít předpřipravené pole čísel, a z každého spočítat jeho odmocninu (a v RTL vašeho Pascalu **není** k dispozici funkce pro výpočet odmocniny). V tomto systému jsme využili API fce `CreateThread` s výše uvedeným prototypem pro zrychlení celého výpočtu rozložením do více vláken. Pokud víme, že proceduru `Process` budeme spouštět jen na `2` jádrovém procesoru, kde každé jádro má taktovací frekvenci `500 MHz`, a každou běžnou instrukci dokončí za `1` takt, OS má nastavený systémový časovač na `10 ms`, a přepnutí kontextu vláken trvá `1000` taktů CPU, tak vysvětlete, zda je níže uvedená implementace vhodná (proveďte kvalifikovaný odhad, zda opravdu povede ke zrychlení oproti jednovláknové implementaci, tj. opakovanému volání `SquareRoot` přímo z **`for`** cyklu z jediného volajícího vlákna `Process`)

```
const MAX = 100000000;
var numbers : array[1..MAX] of longint;
    tids : array[1..MAX] of longword;
procedure Process;
var i : longint;
begin
  for i := 1 to MAX do
    tids[i] := CreateThread(@SquareRoot, i);
  for i := 1 to MAX do
    JoinThread(tids[i]);
end;

procedure SquareRoot(i : longint);
var root, bit, n : longint;
begin
  n := numbers[i]; root := 0; bit := 1 shl 30;
  while bit > n do bit := bit shr 2;
  while bit <> 0 do begin
    if n >= (root + bit) then begin
      n := n - (root + bit);
      root := (root shr 1) + bit;
    end else root := root shr 1;
    bit := bit shr 2;
  end;
  numbers[i] := root;
end;
```

Otázka č. 6 (X)

Detailně vysvětlete, co v uvedeném kontextu u deklarace procedury `ExitThread` znamená klíčové slovo **`cdecl`**. Co vše toto klíčové slovo **`cdecl`** překladači Pascalu přikazuje?

Otázka č. 7

Jak budou v šestnáctkové soustavě vypadat následující čísla níže uvedená v desítkové soustavě, pokud je budeme reprezentovat jako 32-bitová ve dvojkovém doplňku:
(A) `55`
(B) `-55`

Společná část pro otázky označené Y

Předpokládejte níže popsaný CPU vycházející z architektury procesorů Intel 80386 – je to 32-bitový **little-endian** CPU s obecnou registrovou architekturou a s 32-bitovým adresovým prostorem. Procesor má obecné registry `EAX`, `EBX`, `ECX`, `EDX`, `DSI`, `EDI`, `EBP`, registr `ESP` (stack pointer, ukazuje na poslední využitý byte, roste dolů), a registr `EIP`. V instrukční sadě jsou mimo jiné následující instrukce (příznakový registr modifikují pouze aritmetické operace):

- `ADD reg,imm/[addr]/reg` (add without carry)
- `SUB reg,imm/[addr]/reg` (subtract without carry)
- `CALL addr` (direct call), `CALL [addr]` (indirect call)
- `RET imm` (return from subroutine, po dokončení návratu ještě odebere dalších `imm` bytů ze zásobníku)

Všechny výše uvedené instrukce se dvěma operandy mají vždy vlevo cílový a vpravo zdrojový operand. Instrukce mohu mít jednu z následujících variant operandů (povolené varianty viz definice konkrétní instrukce):

- 32-bitový immediate `imm`
- absolutní adresa `[addr]`, kde `[addr]` může být jedno z:
  - `[imm]` adresa daná konstantou
  - `[reg + imm]` adresa daná součtem obsahu registru `reg` a konstanty `imm`
- libovolný registr `reg`

Součástí uvedeného CPU je tzv. FPU (floating-point unit), která dává přímou možnost práce s floating-point čísly typu `double` (64-bitový typ dle IEEE 754), a její část instrukční sady je převzatá z koprocesorů 80387. Tato část instrukční sady CPU využívá variantu zásobníkové architektury (která u tohoto CPU není striktně load/store). CPU má `8` doublových registrů `R0` až `R7` organizovaných do registrového zásobníku – v assembleru ale tyto registry mají „dynamicky" přidělená jména dle aktuálního stavu registrového zásobníku: poslední zabraný registr (tedy vrchol zásobníku) se označuje jako `ST(0)`, předposlední jako `ST(1)`, atd.
Pro práci s těmito registry slouží následující instrukce:

- `FLD [addr]` (64-bit load), `FLD1` (load constant `1.0`)
- `FSTP [addr]` (64-bit store and pop)
- `FSUBP ST(x),ST(0)` (sub and pop: od `ST(x)` odečte `ST(0)`, a odebere `ST(0)` ze zásobníku → tedy původní `ST(1)` se stane `ST(0)`, `ST(2)` se stane `ST(1)`, atd.)
- `FSUB [addr]` (sub: od `ST(0)` odečte hodnotu z `[addr]` a zanechá změněný `ST(0)` na zásobníku)
- `FADDP`, `FADD` (sčítání), `FMULP`, `FMUL` (násobení)

Otázka č. 8 (Y)

Napište v Pascalu bez použití inline assembleru kód procedury (i s deklarací), která by mohla být běžným překladačem přeložena do níže uvedeného kódu disassemblovaného z adresy `$00F72E58` do assembleru 80386/87 (předpokládejte, že všechny procedury a funkce používají variantu Pascalové volací konvence, kde se argumenty předávají na volacím zásobníku zprava doleva, **argumenty odstraňuje volaný**, volaný **nemusí zachovávat obsah** žádných registrů [ani doublových], návratová hodnota typu `double` se předává **na vrcholu registrového zásobníku**, tj. v `ST(0)`):

```
SUB   ESP,8                FLD   [00F1CD98h]
FLD   [00F1CD80h]          FSUB  [ESP+0Ch]
FLD1                       FLD   [00F1CD90h]
FADDP ST(1),ST(0)          FMUL  [00F1CD80h]
FLD   [00F1CD88h]          FADDP ST(1),ST(0)
FADD  [00F1CD90h]          FMULP ST(2),ST(0)
SUB   ESP,8                FADDP ST(1),ST(0)
FSTP  [ESP]                FSTP  [00F1CD80h]
FSTP  [ESP+8]              ADD   ESP,8
CALL  054008CCh            RET   8
FLD   [ESP]
```

Poznámka: víme, že na adrese `$054008CC` je v počítači načtený strojový kód funkce, která měla původně v Pascalu následující deklaraci:

```
function Update(i : double) : double;
```

Otázka č. 9 (Y)

Navrhněte jednoduchý (jen s nejdůležitějšími informacemi) formát spustitelného souboru pro moderní OS, který má umožňovat spouštění programů jakožto procesů na daném CPU. Formát hlaviček popište jako Pascalové záznamy.

Otázka č. 10

Předpokládejte následující deklarace (kde typ `longword` je 32-bitový celočíselný bezznaménkový, typ `word` je 16-bitový celočíselný bezznaménkový):

```
type
  PLongword = ^longword;
  PWord = ^word;
procedure Conv(src : PLongword; dst : PWord);
```

Napište ve Free Pascalu implementaci procedury `Conv` tak, aby převedla vstupní textový null-terminated řetězec `src` z kódování UTF-32 **big-endian** do kódování UTF-16 **little-endian** a výsledné znaky UTF-16 LE null-terminated řetězce uložila do paměti na místo, kam ukazuje argument `dst` (předpokládejte, že váš kód poběží **pouze na little-endian platformách**, a že na místě, kam ukazuje proměnná `dst`, je dostatek nevyužité paměti).
Unicode znaky z rozsahu `$010000` až `$10FFFF` se v UTF-16 kódují následujícím způsobem:
(1) Od kódu znaku se odečte hodnota `$010000`, a výsledné 20-bitové číslo se rozdělí na dvě 10-bitové části, které se zakódují dle následujících pravidel.
(2) Nejvyšších 10-bitů 20-bitové hodnoty se uloží do nejnižších 10 bitů prvního 16-bitového surrogate znaku (leží na nižší adrese). Horních 6 bitů první surrogate má být nastaveno na (vlevo je hodnota bitu 15, vpravo bitu 10):
`1101 10`
(3) Nejnižších 10-bitů 20-bitové hodnoty se uloží do nejnižších 10 bitů druhého 16-bitového surrogate znaku (leží na vyšší adrese). Horních 6 bitů druhé surrogate má být nastaveno na (od první surrogate se liší pouze 10. bitem):
`1101 11`