KIV/ZOS CVIČENÍ 09

2024

L. Pešička

INFO K ZÁPOČTOVÉMU TESTU

- Datum: viz Courseware – Podmínky absolvování – Důležitá data a termíny
- Forma: písemný test na 45 minut

- Linuxové znalosti
  - Jak vypíšu řádky s1.txt obsahující řetězec trolejbus?
  - Co dělá echo $? ; echo $# ; echo `date`
  - Jak nastavím práva na s1, aby vlastník a skupina mohli číst a zapisovat a ostatní nic?
  - Jaký je rozdíl mezi > >> 2> 2>> |
- Příklady ze cvičení
  - Použití semaforu (KS, synchronizace – předávání řízení, ProdKonz.), monitoru
  - Grafy procesů (fork, execve), cobegin/coend
- Teoretické znalosti
  - 1. až 8. přednáška (základní pojmy, správa procesů, fs, …)
  - Zvolit správné odpovědi, vysvětlit pojem

VYTVOŘTE MUTEX POMOCÍ MONITORU V PSEUDOKÓDU

```
#define ZAMCENO          1
#define ODEMCENO         0

monitor muj_mutex {

void mylock()     { … dopište kód zde … }
void myunlock() { … dopište kód zde … }

init() { … }
}
```

Může dojít k zablokování, tak potřebuji podmínkovou proměnnou, nad kterou bude blokace.

Také potřebuji stav, zda je odemčeno či zamčeno.

ŘEŠENÍ

```
#define ZAMCENO        1
#define ODEMCENO       0
monitor muj_mutex {

condition cl;
int stav = ODEMCENO;

void mylock() {
   if (stav == ZAMCENO)
      wait(cl);
   stav = ZAMCENO;
}
```

ŘEŠENÍ

```
void myunlock() {
  stav = ODEMCENO;
  signal(cl);
 }
}
```

UKÁZKA: SEMAFOR POMOCÍ MONITORU

```
monitor monsem {
   int sem;
   condition cl;

   void P() {
     if sem > 0
       sem=sem-1;
     else
       wait(cl);
   }
```

```
   void V() {
      if not empty(cl)
        signal(cl);
      else
        sem=sem+1;
   }

   void monIni(int param)
     { sem = param; }

   init {sem = 1;}
   }
```

NEVÝHODA UVEDENÉHO ŘEŠENÍ

potřebujeme funkci **empty()**,

která zjistí, zda je fronta nad podmínkou prázdná

jak bychom se bez funkce empty obešli?

SEMAFOR POMOCÍ MONITORU 2:

```
monitor m2 {

    int sem;
    condition c1;

    void P() {
      if (sem == 0)
         wait(c1);
      sem = sem-1;
    }

    void V() {
       signal(c1);
      sem = sem + 1;
    }
  …
  }
```

operace P() i V() zde vždy manipulují s proměnnou sem

PRODUCENT / KONZUMENT NA 1 POLOŽKU MONITOREM

```
monitor ProdKonz {
   condition c1, c2;
   int zaplneny = 0;                  // 0 prázdný, 1 plný
   int  buffer_na_jedno;              // buffer na 1 položku

   void insert(int parx);             // vloží položku
   int remove();                      // vyjme položku
   ...
}
```

napište funkce insert(), remove()

```
void insert (int parx) {
  if ( zaplneny == 1)
          wait(c1);
  buffer = parx; zaplneny = 1; signal (c2);
}

int remove() {
  int pom;
  if (zaplneny == 0)
                  wait(c2);
   pom = buffer; zaplneny = 0; signal(c1);
   return pom;
}
```

MONITOR JAVA

- **balík java.util.concurrent.locks**

```
Lock zamek = new ReentrantLock();
Condition podminka1 = zamek. newCondition();
Condition podminka2 = zamek. newCondition();
```

- **Nad instancí třídy Condition lze pro uspání volat metody**
  `await(),`
  `awaitUninterruptibly(),awaitNanos(long nanosTimeout)`
- **Pro probuzení lze volat metody**
  `signal(), signalAll()`

UKÁZKA MONITOR S PODMÍNKOU JAVA

```
try
{
        lock.lock();
        //jsme uvnitř monitoru
        if (!podminka)
                podminka1.awaitUninterruptibly();
        else
                podminka1.signal();
}
finally
{
        lock.unlock();
}
```

CAS - FILOZOFIE

- Compare and Swap instrukce

- Jiná filozofie – místo zamykání spoléháme, že nám pod rukama hodnotu proměnné nikdo nezmění
- S **předpokládanou** hodnotou provedeme výpočet
- Pokud nám risk vyšel a předpokládaná hodnota se **nezměnila**, můžeme dosadit vypočtenou hodnotu
- Pokud nevyšel, musíme výpočet **opakovat** pro novou předpokládanou hodnotu
- Není deterministické

CAS

- Instrukce poskytována procesorem
- Provede atomicky
  test hodnoty, pokud je shoda, nastaví na novou hodnotu
  vždy vrátí původní hodnotu

CAS

int CAS (x, ocekavana, nova)

- provede atomicky:
- vrátí původní hodnotu proměnné x
- pokud x == očekávaná, nastaví x na nova

návratovou hodnotou CAS je původní hodnota x

puvodni = CAS(x,ocekavana,nova)

- pokud puvodni == ocekavana
  - povedlo se, nepočítali jsme zbytečně
  - pokud ne, risk nevyšel, musíme počítat znovu

CAS – UKÁZKY V C, JAVĚ

- C

```
ref = __sync_val_compare_and_swap(&criticalNum,
        myCritNum, myIncreaseCritNum);
```

- Java:

```
public synchronized int compareAndSwap(myInteger var, int
                                expectedValue, int newValue) {
        int oldValue = var.getMyIntValue();
        if (oldValue == expectedValue)
        var.setMyIntValue(newValue);
        return oldValue;
}
```

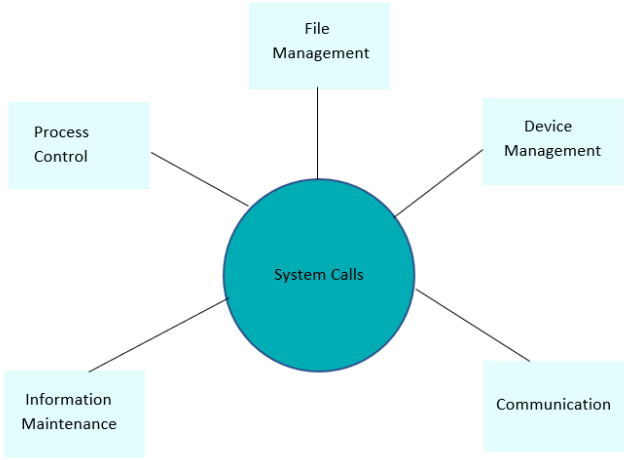
POUŽITÍ CASU (!)

```
znovu:


snimek = x;
nova = nejaky_vypocet(snimek);
puvodni = CAS(x, snimek, nova);
if puvodni == snimek
    printf(“”Uspěli jsme ”);
else
  goto znovu:
```

je zde nedeterminismus, nevíme, kolikrát budeme muset cyklus opakovat

SYSTÉMOVÉ VOLÁNÍ



- Pomocí systémového volání si proces vyžádá službu od jádra operačního systému
- Rozhraní mezi procesem a operačním systémem
- Různé OS mají různá systémová volání
- Předání řízení z neprivilegovaného uživatelského módu do privilegovaného módu jádra

Zdroj obrázku a podrobnější popis jednotlivých skupin:

https://www.geeksforgeeks.org/linux-system-call-in-detail/

SYSTÉMOVÉ VOLÁNÍ

#	Name	Registers			
		eax	ebx	ecx	edx
0	sys_restart_syscall	0x00	-	-	-
1	sys_exit	0x01	int error_code	-	-
2	sys_fork	0x02	struct pt_regs *	-	-
3	sys_read	0x03	unsigned int fd	char __user *buf	size_t count
4	sys_write	0x04	unsigned int fd	const char __user *buf	size_t count
5	sys_open	0x05	const char __user *filename	int flags	int mode
6	sys_close	0x06	unsigned int fd	-	-
7	sys_waitpid	0x07	pid_t pid	int __user *stat_addr	int options
8	sys_creat	0x08	const char __user *pathname	int mode	-

SYSTÉMOVÉ VOLÁNÍ

- Neplést systémové volání a tabulku vektoru přerušení
  - Systémové volání číslo 5 není INT 5 !!

- Pro realizaci systémového volání lze využít softwarového přerušení, v Linuxu konkrétně voláme INT 0x80, tj. podíváme se do tabulky vektorů přerušení na index hexa 0x80 (128 dekadicky) nebo speciální instrukci, např. sysenter

- Toto je jiná tabulka – přehled systémových volání
- Číslo volání, které chceme provést dáme do registru EAX

```
int pid;

int main() {


    __asm__(
          "movl $20, %eax  \n"                  /* getpid system call */
          "int  $0x80      \n"                   /* syscall */
          "movl %eax, pid  \n"                  /* get result */
          );
   printf("Test volani systemove sluzby...\nPID: %d\n", pid);


  return 0;
}
```

```
#include <syscall.h>
#include <unistd.h>
#include <stdio.h>
#include <sys/types.h>
int main(void) {
long ID1, ID2;


/* direct system call */
ID1 = syscall(SYS_getpid);

/* "libc" wrapped system call */
ID2 = getpid();

printf ("syscall(SYS_getpid)=%ld\n", ID1);
printf ("getpid()=%ld\n", ID2);

return(0);
}
```

OBSLUHA PŘERUŠENÍ (!!!)

- I. Mechanismus vyvolání přerušení (vyvolání instrukcí: **INT číslo**)
  - Na zásobník se uloží registr příznaků **FLAGS**
  - Zakáže se přerušení (vynuluje příznak IF – Interrupt Flag v registru FLAGS)
  - Na zásobník se uloží návratová adresa (obsah registrů CS:IP) ukazující na instrukci, **kde budeme** po návratu z přerušení **pokračovat**

- II. Kód obsluhy přerušení – „píše programátor OS“
  - Na zásobník uložíme hodnoty registrů (abychom je procesu nezměnili)
  - Vlastní kód obsluhy (musí být rychlý, případně naplánujeme další věci)
  - Ze zásobníku vybereme hodnoty registrů (aby přerušený proces nic nepoznal)

- III. Návrat z přerušení (instrukce: IRET)
  - Ze zásobníku je vybrána návratová adresa (obsah registrů **CS:IP**) – kde budeme pokračovat
  - Ze zásobníku se obnoví registr FLAGS – obnoví původní stav povolení přerušení

POZNÁMKA - PODROBNĚJI

Po přijetí žádosti o přerušení provádí procesor v reálném režimu tyto akce:

1. do zásobníku se uloží registr příznaků (FLAGS),
2. vynulují se příznaky IF a TF,
3. do zásobníku se uloží registr CS,
4. registr CS se naplní 16bitovým obsahem adresy $n \times 4 + 2$,
5. do zásobníku se uloží registr IP ukazující na neprovedenou instrukci,
6. registr IP se naplní 16bitovým obsahem adresy $n \times 4$.

Výjimky v reálném režimu nevracejí chybový kód. Návrat do přerušeného procesu a jeho pokračování zajistí instrukce IRET, která provede činnosti v tomto pořadí:

1. ze zásobníku obnoví registr IP,
2. ze zásobníku obnoví registr CS,
3. ze zásobníku obnoví příznakový registr (FLAGS).

Zdroj:
M. Brandejs
Mikroprocesory
Intel Pentium

POZNÁMKA - IF

IF (Interrupt Enable Flag) vynulovaný instrukcí CLI zabrání uplatnění vnějších maskovatelných přerušení (generovaných signálem INTR). Nastavení příznaku na jedničku (instrukcí STI) přerušení povoluje. Maskovat lze přerušení od vnějších zařízení (klávesnice, tiskárna atd.), nikoli výjimky, programová přerušení (INT) a NMI (přerušení ze skupiny nemaskovatelných přerušení). Hodnoty CPL a IOPL určují, zda lze tento příznak měnit instrukcemi CLI, STI, POPF, POPFD a IRET.

Zdroj:
M. Brandejs
Mikroprocesory
Intel Pentium

Registr FLAGS procesor u

9. bit je IF

Intel x86 FLAGS register[1]			
Bit #	**Abbreviation**	**Description**	**Category**
FLAGS			
0	CF	Carry flag	Status
1		Reserved	
2	PF	Parity flag	Status
3		Reserved	
4	AF	Adjust flag	Status
5		Reserved	
6	ZF	Zero flag	Status
7	SF	Sign flag	Status
8	TF	Trap flag (single step)	Control
9	IF	Interrupt enable flag	Control
10	DF	Direction flag	Control
11	OF	Overflow flag	Status
12-13	IOPL	I/O privilege level (286+ only), always 1 on 8086 and 186	System
14	NT	Nested task flag (286+ only), always 1 on 8086 and 186	System
15		Reserved, always 1 on 8086 and 186, always 0 on later models	
EFLAGS			
16	RF	Resume flag (386+ only)	System
17	VM	Virtual 8086 mode flag (386+ only)	System
18	AC	Alignment check (486SX+ only)	System
19	VIF	Virtual interrupt flag (Pentium+)	System
20	VIP	Virtual interrupt pending (Pentium+)	System
21	ID	Able to use CPUID instruction (Pentium+)	System

KOMENTÁŘE

- Proč se při obsluze přerušení zakazuje další přerušení?

- Představte si např., že ošetřujete HW přerušení vstup z klávesnice.
- Stisknete jednu klávesu a pustí se obsluha přerušení.
- Pokud by nedošlo k zakázání přerušení, tak než by obsluha přerušení doběhla a byla by stisknuta další klávesa, k čemu by mohlo dojít?
- Původní „rozpracované“ přerušení by bylo přerušeno dalším a to obvykle nechceme. Zpracujeme danou obsluhu a až poté se opět povolí další přerušení (IF flag se nastaví na původní hodnotu).

Příklad mapování vektorů přerušení

Tabulka vektorů přerušení

Kód	Číslo přerušení	Popis
B	INT 00H	Dělení nulou
B	INT 01H	Krokování
B	INT 02H	Nemaskovatelné přerušení
B	INT 03H	Bod přerušení (breakpoint)
B	INT 04H	Přetečení
B	INT 05H	Tisk obrazovky
B	INT 06H	Nesprávný operační kód
B	INT 07H	Není koprocesor
B	INT 08H IRQ0	Přerušení od časovače
B	INT 09H IRQ1	Přerušení od klávesnice
	INT 0aH IRQ2	EGA vertikální zpětný běh
	INT 0bH IRQ3	COM2
	INT 0cH IRQ4	COM1
	INT 0dH IRQ5	Přerušení harddisku
B	INT 0eH IRQ6	Přerušení řadiče disket
	INT 0fH IRQ7	Přerušení tiskárny
B	INT 10H	Služby obrazovky
	INT 11H	Seznam vybavení
	INT 12H	Velikost volné paměti
B	INT 13H	Diskové vstupně-výstupní operace

Příklad možného mapování (původní IBM PC) , může být různé

dva pojmy:

INT … „index“

IRQ … „drát“

všimněte si IRQ0 je zde na INT 08H, na vektoru 08H (tj.od adresy 8*4) bude adresa podprogramu k vykonání dané obsluhy

IRQ – INTERRUPT REQUEST

Nyní nás zajímá HW přerušení

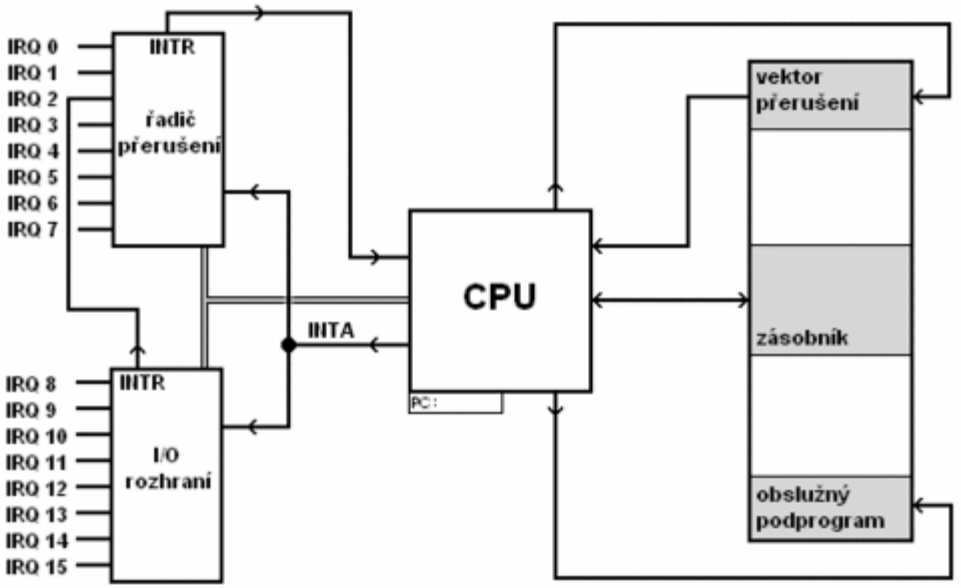
IRQ – signál, kterým zařízení (časovač, klávesnice) žádá procesor o přerušení zpracovávaného procesu za účelem provedení obsluhy požadavku zařízení

NMI – nemaskovatelné přerušení (non-maskable interrupt),
např. nezotavitelná hw chyba

OBSLUHA HW PŘERUŠENÍ

1. zařízení sdělí řadiči přerušení, že potřebuje přerušení
2. řadič upozorní CPU, že jsou čekající (pending) přerušení
3. až je CPU ochotné přijmout přerušení (dokončí rozpracovanou instrukci) tak přeruší výpočet a zeptá se řadiče přerušení, které nejdůležitější čeká a spustí jeho obsluhu
4. uloží stav procesu (návratová adresa, registry,…), provede základní obsluhu zařízení, informuje řadič o dokončení obsluhy, obnoví stav procesu a pokračuje se dále

ŘADIČ PŘERUŠENÍ



2 integrované obvody Intel 8259 (starší

1. spravuje IRQ 0 až 7
   (master, na IRQ2 je připojen druhý)
2. spravuje IRQ 8 až 16

novější systémy Intel APIC Architecture (24 IRQ)

obrázek – zdroj wikipedia

OPAKOVÁNÍ - VYHLADOVĚNÍ

- **Vyhladovění** – stav, kdy jsou vláknu neustále odpírány prostředky. Bez těchto prostředků program nikdy nedokončí svůj úkol.

- Příklad vyhladovění
  - Plánovače SJF, SRT v dávkových systémech
  - Večeřící filozofové – opakovaně zvedají a pokládají levou vidličku, aniž by se najedli

- Plánovače se snaží bránit vyhladovění
  - Počítání přeskočení ve frontě, max. doba čekání ve frontě, epochy, …

OPAKOVÁNÍ - UVÍZNUTÍ

- Uvíznutí (deadlock)
- "Úspěšné dokončení první akce je podmíněno předchozím dokončením druhé akce, přičemž druhá akce může být dokončena až po dokončení první akce."
- Rozdíl mezi uvíznutím a vyhladověním (!)

- Pštros
- Detekce a zotavení
- Předcházení bezpečnou alokací zdrojů
- Strukturální negace Coffman. podmínek

OPAKOVÁNÍ

- Rozdíl dávkový systém – interaktivní – realtimový
  - Využijte pojmy:
  - Deadline, Časové kvantum, úloha spolu se vstupními daty
  - Musí u dávkového systému být známa dopředu přibližná doba výpočtu úlohy?
    U kterého ano, u kterého ne?

- Jaké dvě kategorie realtimových systémů znáte?

- Jaký je rozdíl mezi dávkovým systémem SJF a SRT?

- Vysvětlete pojem časové kvantum.

- Jaký je vztah mezi časovým kvantem a tikem časovače?

OPAKOVÁNÍ

- Co znamená IPC?
- Jaké jsou dvě základní formy IPC?
- Co se stane při stisku CTRL+C?
- Jakým systémovým voláním vytvoříme rouru?
- Jaké výhody a nevýhody má:
  - Dvojí kopírování
  - Randez-vous
- Kdy nemůžeme použít plánování procesů pomocí loterie?
- Jak modifikovat plánovač, aby byl spravedlivý vůči uživateli?

OPAKOVÁNÍ

- Interaktivní systémy používají preemptivní nebo nepreemptivní plánování? Zdůvodněte.
- Co můžeme ovlivnit statickou prioritou?
- K čemu slouží dynamická priorita?
- Vysvětlete pojem rozhodovací mód plánovače.
- Co dělá zero page thread?
- Co znamená CPU burst?
- Vysvětlete pojem nastavení afinity procesu?