Izvedba CppTss

UDŽBENIK POGLAVLJE 4, STRANICE 64-69

Izvedba CppTss-A

- CppTss se razlikuje od konkurentne biblioteke koju podrazumeva međunarodni standard C++11 po tome što ne predviđa da se konkurentni program izvršava **iznad** operativnog sistema, nego da se izvršava **samostalno** (stand alone).
- Izvedba CppTss-a je namenjena za **jednoprocesorski** računar i podrazumeva da nije moguće **lažno** (spurious) aktiviranje niti.
- Izvedba CppTss-a obuhvata **ulazno-izlazne module**, **izvršioca** (**kernel**) i **virtuelnu mašinu**.

Atomski regioni CppTss

- Stvaranje atomskih regiona omogućuje klasa **Atomic_region**.
- Njen **konstruktor** onemogućuje prekide, a **destruktor** vraća prekide u stanje koje je prethodilo akciji konstruktora.

```
{
        Atomic_region ar;
        ...
}
```

Atomski regioni CppTss

- Upotrebu atomskog regiona ilustruje primer **rukovanja pozicijom kursora** u kome nisu moguća štetna preplitanja **niti** i obrada **prekida**, jer telo operacije **get()** klase **Position** obrazuje **atomski region**.
- Pošto se operacija **set()** poziva samo iz **obrada prekida**, njeno telo po definiciji obrazuje **atomski region** (jer su **prekidi onemogućeni u toku obrade prekida,** barem kod nas), pa je tako osigurana međusobna isključivost operacija klase **Position**.

Atomski regioni CppTss

```
class Position {
	int x, y;
public:
	Position();
	void set(int new_x, int new_y);
	void get(int* current_x, int* current_y);
};

Position::Position()
{
	x = 0;
	y = 0;
}
```

Atomski regioni CppTss

```
void Position::set(int new_x, int new_y)
{
	//ne mora atomski jer su prekidi zabranjeni
	x = new_x;
	y = new_y;
}

void Position::get(int* current_x, int* current_y)
{
	Atomic_region ar;
	*current_x = x;
	*current_y = y;
}
```

Klasa Driver

- Pisanje drajvera olakšava klasa **Driver**.
- Nju **nasleđuju** klase koje opisuju ponašanje drajvera.
- Drajvere karakteriše saradnja **obrađivača prekida** i **pozadinskih niti**.
- U okviru klase, koja opisuje ponašanje drajvera, **obrađivač prekida** se predstavlja u obliku **funkcije bez povratne vrednosti** i **bez parametara**.
- Ova funkcija mora biti **static**, da bi se mogla koristiti njena **adresa**.
- Takva moraju biti i sva polja klase kojima ona pristupa.

Klasa Driver

- Operacija **start_interrupt_handling()** klase Driver omogućuje smeštanje **adrese obrađivača prekida** u **tabelu prekida**.
- Prvi argument poziva ove operacije predstavlja **broj vektora prekida**, a drugi **adresu obrađivača prekida**.
- Saradnja **obrađivača prekida** i **pozadinskih niti** podrazumeva da **pozadinske niti čekaju dešavanje spoljnih događaja, a da obrađivači prekida objavljuju dešavanje spoljnih događaja.**

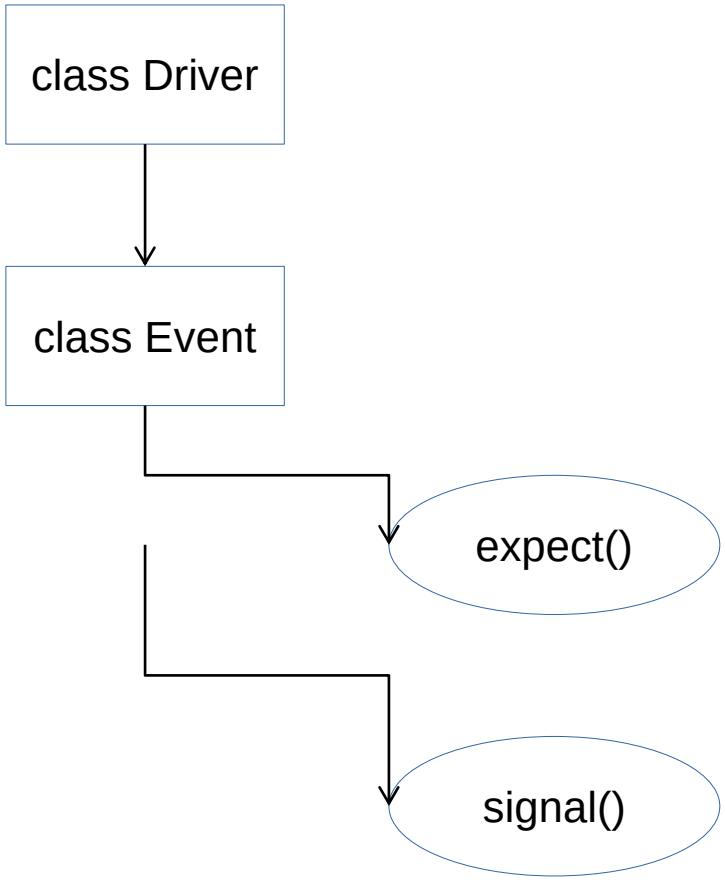
Primer iz stvarne prakse—Linux

```
int request_irq(unsigned int irq,
                irq_handler_t handler,
                unsigned long flags,
                const char *name,
                void *dev)
```

```
typedef irqreturn_t (*irq_handler_t)(int, void *);
```

```
static irqreturn_t intr_handler(int irq, void *dev)
```

Drajveri CppTss-a



- Poziva se iz atomskog regiona
- Nit prelazi u čekanje
- Preključivanje na spremnu nit

- Poziva se iz obrađivača prekida
- Nit koja čeka **najduže** prelazi
- u stanje spremna
- Poziva se na kraju obrade prekida
- samo jednom

Drajver za rukovanje vremenom

- Ovaj drajver registruje proticanje vremena u računaru **brojanjem otkucaja sata**.
- Otkucaji, čiji period zavisi od **takta procesora**, nisu uvek podesni za određivanje vremena u danu, predstavljenog brojem **sati**, **minuta** i **sekundi**, jer sekunda **ne može uvek da se izrazi celim brojem perioda** ovakvih otkucaja.
- Zato je zgodno uvesti **dodatni sat**, čiji **otkucaji** (**prekidi**) imaju period od **tačno jedne sekunde**.
- Ova komponenta se, tradicionalno, zove ’Real Time Clock’ u računarima i vi je imate na matičnoj ploči, tipično zakčenoj za SRAM modul i CR2032 bateriju.

Drajver za rukovanje vremenom

- Rukovanje ovim satom, odnosno rukovanje vremenom opisuje klasa **Timer_driver**.
- Njena tri celobrojna polja: **hour**, **minute** i **second** sadrže broj proteklih **sati**, **minuta** i **sekundi**.
- Početni sadržaj ovih polja određuje **konstruktor** klase **Timer_driver**.

Drajver za rukovanje vremenom

- Pored toga on u **element tabele prekida**, koga indeksira **konstanta TIMER** (**broj vektora prekida dodatnog sata**), smesti **adresu njene operacije interrupt_handler()**, koja je zadužena za **periodičnu izmenu** sadržaja polja **hour**, **minute** i **second**, sa periodom od **jedne sekunde**.

Drajver za rukovanje vremenom

- Klasa **`Timer_driver`** sadrži i operacije **`set()`** i **`get()`** za zadavanje i preuzimanje sadržaja njenih polja.
- To su osetljive operacije, čije **preplitanje sa obradom prekida sata je štetno**.
- Na primer, ako se obrada prekida sata desi **nakon preuzimanja sadržaja polja hour**, **a pre preuzimanja sadržaja polja minute**, i ako je, uz to, broj minuta pre periodične izmene bio **na granici od 59**, tada preuzeto vreme **kasni iza stvarnog za 60 minuta**.
- Da bi se ovakva štetna preplitanja **sprečila**, **preuzimanja** i **zadavanja** sadržaja njenih polja moraju da budu u **atomskim regionima**.

Klasa Timer_driver

```
class Timer_driver : public Driver {
      static int hour;
      static int minute;
      static int second;
      static void interrupt_handler();
public:
      Timer_driver()
      { start_interrupt_handling(TIMER,
        interrupt_handler); };
      void set(const int h, const int m,
                        const int s);
      void get(int* h, int* m, int* s) const;
};
```

Klasa Timer_driver

```
int Timer_driver::hour = 0;
int Timer_driver::minute = 0;
Int Timer_driver::second = 0;

void Timer_driver::interrupt_handler() {
        if(second < 59)
                second++;
        else {
                second = 0;
                if(minute < 59)         minute++;
                else {
                        minute = 0;
                        if(hour < 23) hour++;
                        else hour = 0;
                }
        }
}
```

Klasa Timer_driver

```
void Timer_driver::set(const int h, const int m, const int s)
{
        Atomic_region ar;
        hour = h;
        minute = m;
        second = s;
}

void Timer_driver::get(int* h, int* m, int* s) const
{
        Atomic_region ar;
        *h = hour;
        *m = minute;
        *s = second;
}
```

Klasa Sleep_driver

- Upotrebu klase **Driver** ilustruje i primer drajvera koji omogućuje **uspavljivanje jedne niti** dok ne protekne zadani broj otkucaja sata.
- Ovo uspavljivanje može da se prikaže kao **očekivanje dešavanja zadanog broja otkucaja sata**.
- To opisuje klasa **Sleep_driver**.
- Njena operacija **simple_sleep_for()** omogućuje jednoj niti da **zaustavi svoju aktivnost dok se ne desi zadani broj otkucaja sata**.
- Zadatak **obrađivača prekida** (operacije **interrupt_handler()**) je da **odbroji zadani broj otkucaja sata** i da nakon toga **signalizira** da je moguć nastavak aktivnosti uspavane niti.

Klasa Sleep_driver

```
class Sleep_driver: public Driver {
      static unsigned long countdown;
      static Event alarm;
      static void interrupt_handler();
public:
      Sleep_driver(
       { start_interrupt_handling(TIMER,
                  interrupt_handler); };
      void simple_sleep_for(
                  unsigned long duration);
};

unsigned long Sleep_driver::countdown = 0;
```

Klasa Sleep_driver

```
void Sleep_driver::interrupt_handler()
{
	if((countdown > 0) && ((--countdown) == 0))
		alarm.signal();
}

void Sleep_driver::simple_sleep_for(unsigned long duration)
{
	Atomic_region ar;
	if(duration > 0) {
		countdown = duration;
		alarm.expect();
	};
}
```