PEP Házi dokumentáció

FreeRTOS rész dokumentációja

Piller Botond Ákos

Rusvai Miklós

Feladat leírása

A házi elkészítéséhez felhasználtuk az előző félévben a BAMBI tárgyra írt házi feladatunkat, mely egy torpedó játék volt, ahol egy kurzorral lehetett lépegetni a kijelző szegmensein és a kiválasztott szegmensre lehet lőni. A kijelzőn továbbá egy számláló számolja a lövések számát, hogy hányszor lőttünk a játék során. A lövés során a kijelző bal felső sarkában lévő körön körbe futnak a szegmensek, találat esetén háromszor még villognak, nem találat esetén nem. A kilőtt szegmensek megjelenítve maradnak a kijelzőn.

Task-ok megvalósítása

A programot FreeRTOS-osra úgy írtuk át, hogy létrehoztunk 4 darab task-ot és átírtuk a main-t a gyakorlatokon alkalmazott változatra. A task-okat az app.c fájlban hoztuk létre és itt hívtuk meg a bennük alkalmazott függvényeket. A task.c fájlban inicializáljuk a taskokat, az UART interupt handlert és a kijelzőt is.

Az első task a játék inicializálását végzi el. Ez a task összesen egyszer fut le a játék során és a lefutása után töröljük is a vTaskDelete(NULL) hívással. Ennek a tasknak eggyel nagyobb a prioritása a többi taskhoz képest, hogy mindenképp ez fusson le először, a többi.

```
xSemaphoreTake(guard,portMAX_DELAY);
    szamlalo(lovesszam);
    elrendezes(alapallas);
    kezdes = false;
    xSemaphoreGive(guard);
vTaskDelete(NULL);
```

A második task az UART vételért felel. Ez a task figyeli mikor érkezett az UART-on egy új karakter, amit fel kell dolgozni. Az UART vétel figyelését megszakításosan oldottuk meg. Az UART kezelő task alapesetben alszik, amikor érkezik egy UART interrupt, akkor az felébreszti a task-ot, lefut a task, majd újból elalszik és vár a következő interruptra, hogy felébredjen. Ezen megoldás miatt a játék késleltetése nem nagy. Miután felébredt a task lefoglalja a szemafort, majd a karaktert betölti a data változóba, megvizsgálja az értékét és ez alapján beállítja a többi változó értékét és elengedi a szemafort. A többi változó beállított értéke alapján végeznek műveleteket a további taskok és változtatják a kijelzőt. Az UART task prioritása egyel nagyobb mint a többi meglévő taské, hogy a vétel mindig megtörténjen és ne növelje a játék késleltetését azzal, ha egyen lő prioritás eseten nem ez a task kapja meg a szemafort.

```
vTaskSuspend(NULL);
xSemaphoreTake(guard,portMAX_DELAY);
if(data=='.')
{
win = true;
}
if(data == 'r')
```

```
{
NVIC_SystemReset();
}
```

A harmadik task felel a kijelző írásért, minden olyan függvényt ez a task kezel, amely a kijelzőre ír. Ez a task nem vizsgál feltételt, hanem minden alkalommal, amikor szabad a szemafor le tudjon futni. Amikor megkapja a futási jogot lefoglalja a szemafort, meghívja a kijelzőre író függvényeket, majd mikor minden függvény lefutott elengedi a szemafort.

```
xSemaphoreTake(guard,portMAX_DELAY);
kurzor(x,y);
kijelez(sullyedt,i);
if(win)
{
vege_clear();
vege();
USART_Tx(UART0,'x');
__enable_irq();
}
```

Negyedik task a lövés funkciót valósítja meg. Ez a task figyeli, hogy a loves2 változó igaz-e, ha igen lefoglalja a szemafort, meghívja a lövés művelethez tartozó függvényeket, majd elengedi a szemafort és várakozik. A fejlesztő környezetben hiába végeztük a fordítást debug módban, a fordító így is kioptimalizálta a kódot és a kezdőértékkel inicializált változóknak nem adta az új értéküket, ezért nem értékelődtek ki az if() feltételek és így sose lépett be a negyedik task-ba a program. Ennek kiküszöbölésére az if() feltétel kiértékelését kritikus szakaszba tettük és tiltottuk erre a műveletre a megszakításokat, majd később a kiértékelés eredményét használtuk már csak fel. Ezzel ki tudtuk küszöbölni a problémát.

```
__disable_irq();
bool ez = loves2==true;
__enable_irq();
if(ez){
```