Görevci by Kozmotronik

v2.1.0

Oluşturan Doxygen 1.9.6

Bölüm 1

Görevci Belgelendirmesi

Yazar

İsmail Sahillioğlu (Kozmotronik)

1.0.0.1 Görevciyi Edinme

Görevciyi edinmenin en kolay yolu, Görevciyi Githubdan clon etmek, sonra (varsa) uygulamak istediğiniz portun örnek projesini açıp gerekli değişiklikleri yaparak kullanmaktır. Alternatif olarak kaynak kodunu doğrudan indirip elle projenize dahil edebilirsiniz.

1.0.0.2 Kip Seçimi

Görevci iki farklı kipte kullanılabilir:

- Hafif kip
- Normal kip

Kip, projeye eklenecek *gorevciypl.h* adında bir başlık dosyası içerisinde `grvCALISMA_KIPI` tanımlanarak seçilir. `grvCALISMA_KIPI` 0 olarak tanımlanırsa Görevci *Hafif kip*, 1 olarak tanımlanırsa *Normal kip* için yapılandırılacaktır. Kip olarak *Hafif kip* kullanıldığı durumda programcı, `main` işlevi içinde görev yönetimini kendisi yapar. Her iki kipin kullanımı port demolarında örneklendirilecektir.

1.0.0.3 Görevler Oluşturma

Bir görevin normal bir C işlevinden pek bir farkı yoktur, yalnızca biraz daha yapılandırılmıştır ve sürekli çalışması gereken görevler içlerinde bir sonsuz döngü içerir. Bir görevin temel yapısı şu şekilde olmalıdır:

```
char gorev(gorevTutucu_t tutucu) {
    // Görev kapsamında (scope) kullanılacak değişkenler burada tanımlanabilir.
    // Değerini koruması gereken değişkenler "static" niteleyicisiyle
    // tanımlanmalıdır.
    static char karakter;

    // Bir görev ana görev döngüsünden hemen önce her zaman görev yapısına
    // başvuru olarak parametre alan grvBASLA() ile başlamalıdır.
    grvBASLA(tutucu);

    // Buraya bir kereye mahsus çalışacak kodlar. Örneğin bir giriş - çıkış
    // portunu ilkleme veya bir analog ucunu ilkleme kodları gibi. Buradaki
    // kodlar görevin yaşam süresi boyunca yalnızca bir kez çalışacağı için
    // ilklendirme işlemlerini yapmak için idealdir.

    // Bu döngü bir görev bloğunun ana döngüsüdür. for(;;) biçiminde de
    // yazılabilir. Burada sonsuz döngüde kalmalı, kesinlikle break veya return
    // ile döngüden çıkılmamalıdır.
    while(1) {
        // Buraya görev kodları ve bloklayıcı API çağrıları
    }

    // Tüm görev işlevleri görev yapısına başvuru olarak parametre alan
    // grvBITIR() ile sonlanmalıdır. Akış sonsuz döngüden buraya gelmesi
    // görevin bir daha çalışmamasına neden olabilir.
    grvBITIR(tutucu);
}
```

1.0.0.4 Görevleri Yönetme

Bir görevin bir olay beklemesi ya da gecikmesi gerekiyorsa CPU' yu boşuna meşgul etmemek, gerektiğinde beklemesi gereken görevi bloklayıp çalışmaya hazır başka bir görevin çalışmasını sağlamak Görevcinin temel amaçlarındandır. Ancak bu mekanizmanın düzgün bir şekilde işlemesi için görev bloğu içerisinde, klasik super-loop veya state-machine programlama mantığında kullanılan `return` ve `break` gibi dönüş ve döngü kırma komutları kesinlikle kullanılmamalıdır. Böyle yapmak görevin düzgün ve beklendiği gibi çalışmamasına neden olur. Bunun yerine aşağıdaki örnek durumlara uygun düşen API' ler kullanılmalıdır.

1.0.0.4.1 Bir görevi geciktirme

- `grvGECIK_MS()` - *işletim sistemlerinde* `sleep()` *işlevlerine benzer*
- `grvKOSULLU_GECIK_MS()` - *işletim sistemlerinde* `sleep()` *işlevlerine benzer*

1.0.0.4.2 Bir koşul veya olayın beklenmesi

- `grvKOSUL_BEKLE()`
- `grvBU_KOSULDA_BEKLE()`

1.0.0.4.3 Verilere erişimde senkronizasyon

- `grvBAYRAK_BEKLE()` - *işletim sistemlerindeki* `wait()` *işlevlerine benzer*
- `grvBAYRAK_IMLE()` - *işletim sistemlerindeki* `signal()` *işlevlerine benzer*

1.0.0.4.4 CPU kontrolünden gönüllü olarak vazgeçme Bazen bir görev kendi isteğiyle kontrolü çalışmak için bekleyen başka bir göreve vermek isteyebilir. Böyle bir durumda:

- `grvVAZGEC()` - *işletim sistemlerindeki* `yield()` *işlevlerine benzer*
- `grvKOSULA_DEK_VAZGEC()`

Bölüm 2

Lisans

Yazar

İsmail Sahillioğlu (Kozmotronik)

Copyright

MIT Lisansı.

2.0.1 MIT Lisansı / License

2.0.1.1 Türkçe Sürümü

Görevci projesi, metni aşağıda verilen MIT açık kaynak lisansı altında yayınlanmaktadır. Bu lisans gorevci projesindeki kaynak kodu dosyalarının hepsini kapsamaktadır. Ayrıca demo projeleri bu projenin parçası olmayan üçüncü parti yazılımlar içerebilir ve bu yazılımlar bu projeden ayrı olarak lisanslanmış olabilir.

2.0.1.2 İngilizce Sürümü

Bölüm 3

Modül İndeksi

3.1 Modüller

Tüm modüllerin listesi aşağıdadır:

Bölüm 4

Veri Yapıları İndeksi

4.1 Veri Yapıları

Kısa tanımlarıyla birlikte veri yapıları:

Bölüm 5

Dosya İndeksi

5.1 Dosya Listesi

Bu liste tüm dokümante edilmiş dosyaları kısa açıklamalarıyla göstermektedir:

Bölüm 6

Modül Dokümantasyonu

6.1 Senkronizasyon

Modüller

- Bayrak

6.1.1 Ayrıntılı tanımlama

Birbirinden bağımsız çalışan görevlerin paylaşılan kaynaklara erişimini senkronize etmeye yarayan mekanizmalar.

6.2 Veri Yönetimi

Modüller

- Unsigned Char Kuyruk Yönetimi

6.2.1 Ayrıntılı tanımlama

Veri yapıları kullanarak veri organizasyonunu kolaylaştıracak mekanizmalar.

6.3 Port - Hedef Platform Yönetimi

Modüller

- MPLAB XC8 PIC18 Portu

6.3.1 Ayrıntılı tanımlama

Görevci temel olarak C dili ile yazıldığı için hemen hemen her platformla uyumludur ancak yine de platforma özel olabilecek bazı işlevselliklere bağımlıdır. Örneğin Görevci'ye ilgili platform tarafından bir sistem tiki zamanlayıcısı sunulmalıdır. Ancak herbir hedef platformun zamanlayıcı ilkleme ve yönetim işlevleri kendine özel olduğundan Görevci bu aşamada ilgili platforma arayüzlenmelidir veya diğer bir deyişle bağımlı bölümler platformun sunduğu parçalar ile birleştirilmelidir. Ancak bu, platform çeşitliliğinden dolayı yönetmesi zor bir işlemdir. Neyseki C dili modüler bir yapıya sahip olduğundan bu farklı hedef platformlara entegrasyon daha kolay yollarla sağlanabilmektedir. Bu yol, herbir hedef platforma özel temel tanımlamaları yapıp, gereken bazı işlevselliklerin gerçeklemesini Görevci kütüphanesini kullanacak olan geliştiriciye bırakmaktır.

6.4 Bayrak

Veri Yapıları

- struct Bayrak

Makrolar

- #define grvBAYRAK_ILKLE(b, s)
- #define grvBAYRAK_BEKLE(g, b)
- #define grvBAYRAK_IMLE(b)

Typedef'ler

- typedef Bayrak **bayrak_t**

  Kolaylık sağlamak için Bayrak tür tanımlaması.

6.4.1 Ayrıntılı tanımlama

Bu modül sayıcı bayrakları görevlerin üzerinde gerçekler. Bayraklar birkaç işlem sağlayan bir senkronizasyon ilkelleridir: **bekle** ve **imle** (wait and signal). Bekleme işlemi bayrak sayıcısını yoklar ve sıfırsa görevi bloklar. İmleme işlemi bayrak sayıcısını arttırır ama görevi bloklamaz. Başka bir görev imlenen bayrağı beklemek için bloklanmışsa, bloklanan görev, yeniden çalışabilir olacaktır.

Bayraklar, izleyiciler, ileti tamponları ve bağlı tamponlar gibi daha yapılandırılmış senkronizasyon ilkellerini gerçeklemek için kullanılabilirler.

6.4.2 Makro Dokümantasyonu

6.4.2.1 grvBAYRAK_BEKLE

```
#define grvBAYRAK_BEKLE(
              g,
              b )
```

Bir bayrağı bekler.

Koşut (paralel) çalışan birden fazla görevden bir kaynağa erişirken; bayrak ilklenirken belirlenen erişim sayısına bağlı olarak erişimi ilk alan görev veya görevler kaynağa veya veriye erişim sağlarken, erişim sınırı dolduktan sonra erişen görev veya görevler bloklanacak, ilk erişim alan görevler erişimi grvBAYRAK_IMLE() ile saldıktan sonra bloklanan görevler artan erişim sınırı kadar kaynağa veya veriye erişebilecektir. Kullanımı basitçe şöyledir:

```
bayrak_t bayrak;
char korunanKaynak;
//...
// Erişimi almaya çalış; alamazsa bloklanır.
grvBAYRAK_BEKLE(gorevTutucu, &bayrak);
// Kaynağa erişim sağlandı, artık değiştirilebilir.
korunanKaynak = 'A';
// İşlem bitince diğer görevlerin erişebilmesi için erişimi sal.
grvBAYRAK_IMLE(gorevTutucu, &bayrak);
```

Parametreler

g	Bayrağı alan görevin tutucusu.
b	Bayrağa referans.

6.4.2.2 grvBAYRAK_ILKLE

```
#define grvBAYRAK_ILKLE(
              b,
              s )
```

Bir bayrağı ilkler.

Bayraklar kullanılmadan önce ilklenmelidirler. İlklenmeyen bayraklar belirsiz veya beklenmeyen sonuçlara neden olabilir. Bir bayrak basitçe şöyle ilklenir:

```
bayrak_t bayrak;
//...
// main döngüsünde veya uygun bir yerde ilkleme yapılır.
grvBAYRAK_ILKLE(&bayrak, 1);
```

Parametreler

b	Bayrağa referans.
s	Sayım, yani bir kaynağa aynı anda erişebilecek görev sayısı.

6.4.2.3 grvBAYRAK_IMLE

```
#define grvBAYRAK_IMLE(
              b )
```

Bir bayrağı imler.

Birçok işletim sistemindeki signal veya give gibi işlevlere benzer. Bir görev eriştiği kaynakla işlem yapmayı bitirince, başka görevlerin de bu kaynağa erişebilmesi için bu makroyu çalıştırarak erişimi salar. Kullanımı basitçe şöyledir:

```
bayrak_t bayrak;
char korunanKaynak;
//...
// Erişimi almaya çalış; alamazsa bloklanır.
grvBAYRAK_BEKLE(gorevTutucu, &bayrak);
// Kaynağa erişim sağlandı, artık değiştirilebilir.
korunanKaynak = 'A';
// İşlem bitince diğer görevlerin erişebilmesi için erişimi sal.
grvBAYRAK_IMLE(gorevTutucu, &bayrak);
```

Parametreler

b	Bayrağa referans.

6.5 Görev Yönetimi

Dosyalar

- dosya gorev.c

Veri Yapıları

- struct GorevKontrolBlogu
- struct SureKontrolBlogu

Makrolar

- #define NULL ((void∗) 0)
- #define **grvCALISMA_KIPI** 1

  Görev çalışma kipi tanımlanmamışsa varsayılan normal kip.
- #define **grvTIK_SURESI_uS** 1000u

  Varsayılan tik süresi 1 ms.
- #define **grvTIK_SURESI_MS** (grvTIK_SURESI_uS / 1000u)

  Tik süresinin milisaniye türünden değeri.
- #define grvILKLE(g)
- #define grvBASLA(g)
- #define grvBITIR(g)
- #define grvGECIK_MS(g, s, gecikme)
- #define grvKOSULLU_GECIK_MS(g, s, gecikme, kosul)
- #define grvKOSUL_BEKLE(g, kosul)
- #define grvBU_KOSULDA_BEKLE(g, kosul)
- #define grvSIFIRLA(g)
- #define grvCIK(g)
- #define grvVAZGEC(g)
- #define grvKOSULA_DEK_VAZGEC(g, kosul)

Typedef'ler

- typedef void ∗ **gorevTutucu_t**

  Görevin işlevinin kendisine referansı (handle for function).
- typedef char(∗ **is_t**) (gorevTutucu_t)

  Görevin işlev türü (function pointer type).
- typedef struct GorevKontrolBlogu **gorev_t**

  Kolaylık sağlamak için GorevKontrolBlogu tür tanımı.
- typedef gorev_t ∗ **pgkb_t**

  GKB için referans türü (pointer).
- typedef struct SureKontrolBlogu **sure_t**

  Kolaylık sağlamak için SureKontrolBlogu tür tanımı.

Fonksiyonlar

- unsigned int grvTikSayimi (void)
- void grvTikKesmeIsleyici (void)
- pgkb_t grvOlustur (is_t is)
- void grvGorevciyiBaslat (void)
- pgkb_t grvKimlikIleGorevBlogunuAl (const unsigned char kimlik)
- void grvBaslat (const unsigned char kimlik)
- void grvDurdur (const unsigned char kimlik)

Süre Birimi Dönüştürücüleri

Mikrosaniye <-> tik; milisaniye <-> tik süre dönüşümleri

- #define **grvuS_TIK_CEVIR**(us) (us / grvTIK_SURESI_uS)

  Mikrosaniye süre değerini tik süre değerine dönüştürür.
- #define **grvMS_TIK_CEVIR**(ms) (ms / grvTIK_SURESI_MS)

  Milisaniye süre değerini tik süre değerine dönüştürür.
- #define **grvTIK_uS_CEVIR**(tik) (tik ∗ grvTIK_SURESI_uS)

  Tik süre değerini mikrosaniye süre değerine dönüştürür.
- #define **grvTIK_MS_CEVIR**(tik) (tik ∗ grvTIK_SURESI_MS)

  Tik süre değerini milisaniye süre değerine dönüştürür.

Görev Durum Kodları

Görevlerin işletimdeki durumlarını belirten kodlardır.

- #define **grvBEKLIYOR** 0

  Görev bir olayın gerçekleşmesini bekliyor.
- #define **grvCIKTI** 1

  Görev işlemin herhangi bir noktasında çıktı (çalışmasını sonlandırdı).
- #define **grvBITTI** 2

  Görev işlemi tamamlayarak bitti.
- #define **grvVAZGECTI** 3

  Görev işlemin herhangi bir noktasında gönüllü olarak vazgeçti.

6.5.1 Ayrıntılı tanımlama

Görevlerin yönetimini gerçekleştiren API.

6.5.2 Makro Dokümantasyonu

6.5.2.1 grvBASLA

```
#define grvBASLA(
              g )
```

Görev başlangıç noktası.

Görevlerin içinde sonsuz döngüye girmeden önce çağrılmalıdır. Bu yapılmazsa derleme hatalarıyla karşılaşılabilir veya görev çalışmaz. Her görevin parametresi olan gorevTutucu_t parametresi verilerek çağrılır:
```
grvBASLA(tutucu);
```

Parametreler

g	gorevTutucu_t türünde bir tutucu (handle) alır. Bu, görev kontrol bloğuna bir başvurudur.

6.5.2.2 grvBITIR

```
#define grvBITIR(
              g )
```

Görev bitiş noktası.

Görevlerin içinde sonsuz döngünün bitiminden sonra çağrılmalıdır. Bu yapılmazsa derleme hatalarıyla karşılaşılabilir veya görev çalışmaz. Her görevin parametresi olan gorevTutucu_t parametresi verilerek çağrılır:
```
grvBITIR(tutucu);
```

Parametreler

g	gorevTutucu_t türünde bir tutucu (handle) alır. Bu, görev kontrol bloğuna bir başvurudur.

6.5.2.3 grvBU_KOSULDA_BEKLE

```
#define grvBU_KOSULDA_BEKLE(
              g,
              kosul )
```

Koşul sağlanıyorken görevi bloklar.

Verilen bir koşul sağlandığı sürece görevi bloklar. Koşul başka bir görev tarafından değiştirilip artık sağlanmadığında bloklanan görev kaldığı yerden çalışmayı sürdürür. Örneğin bir kuyruk API'si olduğunu varsayalım ve aşağıdaki işlev prototipi basitçe kuyruk doluysa true değilse false döndürüyor.

```
bool kuyrukDolumu(void)
```

Bu durumda kuyrukDolumu() işlevi false döndürene dek, yani kuyruk boşalana dek görev bloklanacaktır.

```
grvBU_KOSULDA_BEKLE(gorevTutucu, kuyrukDolumu());
```

Parametreler

g	gorevTutucu_t türünde bir tutucu (handle) alır. Bu, görev kontrol bloğuna bir başvurudur.
kosul	O anda sağlanan ve sağlanmaması için beklenecek bir koşul belirten ifade.

6.5.2.4 grvCIK

```
#define grvCIK(
               g )
```

Bir görevden çıkar.

Bir kez çıkış yapıldı mı artık bu görev çalıştırılmak için çağrılmaz. Yeniden çağrılması için görevin gorevOlustur() API'si ile yeniden çalışacak görevler listesine eklenmesi gerekir. Her görevin parametresi olan gorevTutucu_t parametresi ve koşul ifadesi verilerek çağrılır:

```
grvCIK(gorevTutucu);
```

Parametreler

g	gorevTutucu_t türünde bir tutucu (handle) alır. Bu, görev kontrol bloğuna bir başvurudur.

6.5.2.5 grvGECIK_MS

```
#define grvGECIK_MS(
               g,
               s,
               gecikme )
```

Bir görevi verilen milisaniye gecikme süresi kadar geciktirir.

Görevlerin içinde görevin ertelenmesi, geciktirilmesi veya belli bir süre sonra çalıştırılması gereken durumlarda çağrılır. Bu makro çağrıldığında görev, verilen süre kadar bloklanır (artık çalışmaz). Her görevin parametresi olan gorevTutucu_t parametresi, süre takibini yapmak için kullanılacak sure_t türünden değişkenin başvurusu ve milisaniye türünden gecikme süresi verilerek çağrılır:

```
static sure_t sureTutucu;
// ...
grvGECIK_MS(tutucu, &sureTutucu, 1000);
```

Parametreler

g	gorevTutucu_t türünde bir tutucu (handle) alır. Bu, görev kontrol bloğuna bir başvurudur.
s	sure_t türünde bir süre tutucu değişkene başvuru.
gecikme	Milisaniye türünden gecikme süresi.

6.5.2.6 grvILKLE

```
#define grvILKLE(
              g )
```

Görevleri ilkler, ilk kullanım için hazırlar.

Görevler çalışmaya başlamadan önce bu makro ile ilklenmelidir. Bu yapılmazsa görevler beklendiği gibi çalışmayabilir. Dahili olarak Görevci tarafından kullanılır. Uygulama katmanından direk çağrılmamalıdır.

Parametreler

g	gorevTutucu_t türünde bir tutucu (handle) alır. Bu, görev kontrol bloğuna bir başvurudur.

6.5.2.7 grvKOSUL_BEKLE

```
#define grvKOSUL_BEKLE(
              g,
              kosul )
```

Koşul gerçekleşene dek görevi bloklar.

Her görevin parametresi olan gorevTutucu_t parametresi ve koşul ifadesi verilerek çağrılır:

```
int miktar = 0;
//...
grvKOSUL_BEKLE(tutucu, miktar > 0);
```

Parametreler

g	gorevTutucu_t türünde bir tutucu (handle) alır. Bu, görev kontrol bloğuna bir başvurudur.
kosul	Gerçekleşmesi beklenecek bir koşul belirten ifade.

6.5.2.8 grvKOSULA_DEK_VAZGEC

```
#define grvKOSULA_DEK_VAZGEC(
              g,
              kosul )
```

Bir görevin sırasını bir koşul sağlanana dek diğer görevlere vermek için kullanılır.

Bir görev yaptığı işlemde belli bir aşamaya gelip bir koşul beklemesi gerekiyorsa grvKOSUL_BEKLE'ye alternatif olarak bu makroyu kullanabilir. Bu durumda görev; belirtilen koşul sağlanana dek CPU kontrolünü diğer görevlere gönüllü olarak verecektir. Her görevin parametresi olan gorevTutucu_t parametresi ve koşul ifadesi verilerek çağrılır:

```
bool mesajAlindi; // Global bir değişken
//...
grvKOSULA_DEK_VAZGEC(gorevTutucu, mesajAlindi == true);
```

Parametreler

g	gorevTutucu_t türünde bir tutucu (handle) alır. Bu, görev kontrol bloğuna bir başvurudur.
kosul	Sağlanması gereken koşul ifadesi.

6.5.2.9 grvKOSULLU_GECIK_MS

```
#define grvKOSULLU_GECIK_MS(
              g,
              s,
              gecikme,
              kosul )
```

Bir görevi koşullu geciktirir.

Görevlerin içinde bir koşula bağlı olarak görevin ertelenmesi, geciktirilmesi veya belli bir süre sonra çalıştırılması gereken durumlarda çağrılır. Bu makro çağrıldığında belirtilen koşul sağlandıkça görev, verilen süre kadar bloklanır (artık çalışmaz). Herhangi bir gecikme aşamasında koşulun sağlanmadığı saptanır saptanmaz gecikme iptal edilir ve görev kaldığı yerden çalışmayı sürdürür.

Her görevin parametresi olan gorevTutucu_t parametresi, süre takibini yapmak için kullanılacak sure_t türünden değişkenin başvurusu, milisaniye türünden gecikme süresi ve son olarak gecikmenin bağlı çalışacağı koşul verilerek çağrılır:

```
static sure_t sureTutucu;
static int a;
// ...
// a değişkeni 0'dan farklı bir değer aldığında gecikme hangi
// aşamada olursa olsun görev, bekleme durumundan çıkıp
// çalışma durumuna geçecek ve kaldığı yerden devam edecektir.
grvKOSULLU_GECIK_MS(tutucu, &sureTutucu, 1000, a == 0);
```

Parametreler

g	gorevTutucu_t türünde bir tutucu (handle) alır. Bu, görev kontrol bloğuna bir başvurudur.
s	sure_t türünde bir süre tutucu değişkene başvuru.
gecikme	Milisaniye türünden gecikme süresi.
kosul	Gecikmenin bağlı çalışacağı koşul.

6.5.2.10 grvSIFIRLA

```
#define grvSIFIRLA(
              g )
```

Bir görevi sıfırlar (yeniden başlatır).

Görevi sıfırlar (yeniden başlatır). Dolayısıyla görev bir sonraki çalışmada kaldığı yerden değil, en baştan çalışmaya başlar. Her görevin parametresi olan gorevTutucu_t parametresi ve koşul ifadesi verilerek çağrılır:

```
grvSIFIRLA(gorevTutucu);
```

Parametreler

g	gorevTutucu_t türünde bir tutucu (handle) alır. Bu, görev kontrol bloğuna bir başvurudur.

6.5.2.11 grvVAZGEC

```
#define grvVAZGEC(
              g )
```

Bir görevin sırasını başka bir göreve vermek için kullanılır.

Bazen bir görev bir işlemi tamamlayıp başka önemli zaman kısıtı olan bir işlem yapmayacaksa, bu makroyu çalıştırarak CPU kontrolünü çalışmak için bekleyen diğer görevlere gönüllü olarak verir. Bu makro diğer sistemlerdeki yield işlevine benzer çalışır. Her görevin parametresi olan gorevTutucu_t parametresi verilerek çağrılır:

```
grvVAZGEC(gorevTutucu);
```

Parametreler

g	gorevTutucu_t türünde bir tutucu (handle) alır. Bu, görev kontrol bloğuna bir başvurudur.

6.5.2.12 NULL

```
#define NULL ( (void*) 0 )
```

include sorunlarına önlem olarak NULL tanımlaması.

6.5.3 Fonksiyon Dokümantasyonu

6.5.3.1 grvBaslat()

```
void grvBaslat (
             const unsigned char kimlik )
```

Kimliği verilen görevi uyandırır / çalıştırır.

Verilen kimliğe sahip GKB bulunamazsa işlem sürdürülmez.

Parametreler

in	*kimlik*	GKB kimliği.

6.5.3.2 grvDurdur()

```
void grvDurdur (
            const unsigned char kimlik )
```

Kimliği verilen görevi uyutur / durdurur.

Verilen kimliğe sahip GKB bulunamazsa işlem sürdürülmez.

Parametreler

in	*kimlik*	GKB kimliği.

6.5.3.3 grvGorevciyiBaslat()

```
void grvGorevciyiBaslat (
            void  )
```

Görevlerin çalışacağı sonsuz döngüyü başlatır.

Listedeki herbir görev sırasıyla çalıştırılır. Görevler işbirlikçi (cooperative) olarak çalışırlar, çalışan göreve, kendisi kontrolü teslim edinceye dek müdahale edilmez.

Uyarı

Bu işlev main işlevinden doğrudan çağrılmamalıdır. Bunun yerine hangi platform için geliştirme yapılıyorsa, o platformun portGorevciyiBaslat() işlevi ile görevci başlatılmalıdır. portGorevciyiBaslat() işlevi platforma özel gereken ilklemeleri yapacak ve ardından grvGorevciyiBaslat işlevini çağıracaktır. Doğru uygulama bu şekildedir. Aksi takdirde beklenmedik sonuçlar elde edilebilir.

Ayrıca Bakınız

portGorevciyiBaslat().

6.5.3.4 grvKimlikIleGorevBlogunuAl()

```
pgkb_t grvKimlikIleGorevBlogunuAl (
            const unsigned char kimlik )
```

Kimliği verilen görev kontrol bloğuna referans döndürür.

Parametreler

in	*kimlik*	GKB kimliği.

Döndürdüğü değer

gorev_t türünde görev kontrol bloğuna referans, yoksa NULL.

6.5.3.5 grvOlustur()

```
pgkb_t grvOlustur (
            is_t is )
```

Görev işlevini alıp bir Görev Kontrol Bloğu (GKB) oluşturur.

Görev olarak tanımlanmış işlev referansını alıp yeni bir görev olarak görevler listesine ekler. Görevler uygulamanın main bölümünde bu işlev kullanılarak oluşturulmalıdır.

Parametreler

in	*is*	İşi yapacak görev kodu bloğu (bir function pointer).

Döndürdüğü değer

gorev_t türünde görev kontrol bloğuna referans, yoksa NULL.

6.5.3.6 grvTikKesmeIsleyici()

```
void grvTikKesmeIsleyici (
            void  )
```

Sistem tik işleyicisi.

Görevlerin süre takibi yapabilmesi için bu işlevin bir sistem tik timer kesmesi tarafından çağrılması gereklidir.

Bu işlev sistem sürevcisi kesmesinden çağrılır.

6.5.3.7 grvTikSayimi()

```
unsigned int grvTikSayimi (
            void  )
```

Anlık sistem tiki değerini verir.

Normalde Görevci tarafından süre takibi yapmak için kullanılır. Nitekim istendiği takdirde bu başlık dosyasını ekleme koşulu ile uygulama içinden de anlık sistem tiki sayımını almak için kullanılabilir. Sistem tiki sayımını atomik olarak okuyup döndürür. Sistem tiki geçen sürenin milisaniye türünden değeri değil, gorevciypl.h dosyasında tanımlanan SISTEM_TIK_SURESI_uS süre değerinin kaç kez oluştuğunu sayan bir birimdir. Diğer bir deyişle sistemin kalp atışının sayısıdır. Ancak mikrosaniye ve milisaniye birimlerine kolayca dönüştürülebilir.

Döndürdüğü değer

Anlık sistem tiki değeri.

6.6 Unsigned Char Kuyruk Yönetimi

Dosyalar

- dosya uckuyruk.c

Veri Yapıları

- struct UCKuyruk

Makrolar

- #define NULL ((void∗) 0)

Typedef'ler

- typedef struct UCKuyruk **uckuyruk_t**

  Kolaylık sağlaması için UCKuyruk tür tanımı.
- typedef uckuyruk_t ∗ **puck_t**

  uckuyruk_t yapısı için başvuru türü (okunaklılığı iyileştirmek için)

Fonksiyonlar

- void uckuyrukIlkle (const unsigned char kapasite, puck_t kuyruk, unsigned char ∗tampon)
- char uckuyrukKuyrukla (puck_t kuyruk, const unsigned char deger)
- unsigned char uckuyrukCokluKuyrukla (puck_t kuyruk, const unsigned char ∗kaynak, const unsigned char nicelik)
- unsigned char uckuyrukBastakiOge (puck_t kuyruk)
- unsigned char uckuyrukKuyruktanAl (puck_t kuyruk)
- unsigned char uckuyrukCokluAl (puck_t kuyruk, unsigned char ∗hedef, const unsigned char nicelik)
- char uckuyrukKuyrugaAktar (puck_t kaynak, puck_t hedef)
- unsigned char uckuyrukKuyrugaCokluAktar (puck_t kaynak, puck_t hedef, const unsigned char nicelik)
- char uckuyrukKuyrugaKopyala (puck_t kaynak, puck_t hedef)
- unsigned char uckuyrukKuyrugaCokluKopyala (puck_t kaynak, puck_t hedef, const unsigned char nicelik)
- void uckuyrukBosalt (puck_t kuyruk)
- void uckuyrukNBosalt (puck_t kuyruk, unsigned char n)
- unsigned char uckuyrukDoldur (puck_t kuyruk, const unsigned char deger, const unsigned char nicelik)
- unsigned char uckuyrukOgeSayimi (puck_t kuyruk)
- char uckuyrukDolu (puck_t kuyruk)
- unsigned char uckuyrukKalanKapasite (puck_t kuyruk)
- char uckuyrukBos (puck_t kuyruk)

6.6.1 Ayrıntılı tanımlama

Görevci için ardışıl (serial) FIFO modelinde bir kuyruk kütüphanesi.

uckuyruk unsigned char kuyruk kısaltması olarak kullanılmıştır. Bu modülün amacı statik olarak tanımlanacak unsigned char türünde bir dizinin referansını tutup o dizi üzerinde FIFO (İlk giren ilk çıkar) türünde bir kuyruk yönetimi yapmaktır. Modül, uckuyruk_t türünde bir yapıya sınırlı kapasiteye (en fazla 255) sahip statik bir unsigned char dizisi bağlamak koşuluyla, tek bir uygulama içerisinde birden çok FIFO kuyruğu yönetimi yapabilecek şekilde yazılmıştır. Özellikle seri portlar veya LCD ekranlar gibi görece yavaş çalışan donanımlara birden çok ardı ardına veri yazılması gereken durumlarda bu kuyruğu kullanmak çok avantajlıdır. Bu FIFO kuyruğu kullanıldığı takdirde, bu tür yavaş aygıtların veriyi byte-byte göndermesini beklemek yerine kuyruğa yazıp sonra ilgili donanım göndermeye her müsait olduğunda, kuyruktan alınıp ilgili donanıma verilebilir. Böylece CPU zamanı yavaş donanımların işini bitirmesini beklemek için boşuna harcanmaz.

Kullanımı oldukça basittir:

```
// Önce statik bir dizi tanımlanır:
#define DIZI_KAPASITE 48
unsigned char dizi[DIZI_KAPASITE];

// Dizi tanımlandıktan sonra bir uckuyruk_t tanımlanır:
uckuyruk_t kuyruk;

// Tanımlanan kuyruk ilgili parametrelerle ilklenir:
uckuyrukIlkle(DIZI_KAPASITE, &kuyruk, dizi);

// Kuyruk bir kez ilklendi mi artık kullanıma hazırdır:
// Kuyruklamak için:
if(uckuyrukKuyrukla(&kuyruk, 25) == 0) {
     // Kuyruklama başarısız; kuyruk dolu.
     // Burada farklı yeniden deneme stratejileri uygulanabilir.
}
else {
     // Kuyruklama başarılı.
}

// Kuyruktan almak için:
if(uckuyrukBos(&kuyruk)) {
     // Kuyruk boş, alınacak bir şey yok.
     // Burada farklı bekleme stratejileri uygulanabilir.
}
else {
     // Kuyruktan veri var, alınabilir.
     unsigned char veri = uckuyrukKuyruktanAl(&kuyruk);
}
```

Eğer kuyrukta char türünde veriler saklanmak istenirse tek yapılması gereken; veriyi kuyruktan alırken char türüne dönüştürmektir (type casting). Önceki örneği char türü için güncellemek istersek yalnızca kuyruktan alma işleminde değişiklik yapmamız gerekir:

```
// Kuyruklamak için:
if(uckuyrukKuyrukla(&kuyruk, 'A') == 0) {
     // Kuyruklama başarısız; kuyruk dolu.
     // Burada farklı yeniden deneme stratejileri uygulanabilir.
}
else {
     // Kuyruklama başarılı.
}

// Kuyruktan almak için:
if(uckuyrukBos(&kuyruk)) {
     // Kuyruk boş, alınacak bir şey yok.
     // Burada farklı bekleme stratejileri uygulanabilir.
}
else {
     // Char türünde veriyi almak için tek yapılması gereken casting
     // işlemidir.
     char veri = (char) uckuyrukKuyruktanAl(&kuyruk);
}
```

Basit tekli kuyruklama ve kuyruktan almaya ek olarak çoklu kuyruklama ve çoklu kuyruktan alma API'leri de vardır. Bunlar:

- uckuyrukCokluKuyrukla()
- uckuyrukCokluAl() işlevleridir.

Bu kadarla da sınırlı değil! Çeşitli ardışıllaştırma (serializing) teknikleri kullanarak byte ırmağına dönüştürülmüş daha büyük türde ve yapıda veriler de kuyruklanabilir. Aynı kuyruktan veri eski haline döndürmek için; kuyruklamadan önce uygulanan ardışıllaştırma tekniğinin tersi bir teknik ile istenen tür ve yapıdaki veri byte ırmağı halinde kuyruktan alınıp özgün yapısına dönüştürülebilir.

6.6.2 Makro Dokümantasyonu

6.6.2.1 NULL

```
#define NULL ( (void*) 0 )
```

include sorunlarına önlem olarak NULL tanımlaması.

6.6.3 Fonksiyon Dokümantasyonu

6.6.3.1 uckuyrukBastakiOge()

```
unsigned char uckuyrukBastakiOge (
            puck_t kuyruk )
```

Bir kuyruğun en başındaki veriyi gösterir (peek).

Gösterilen veri kuyruktan çıkarılmaz, veriyi kuyruktan çıkarıp almak için uckuyrukKuyruktanAl() işlevini kullanınız. kuyruk parametresi NULL verilirse bu işlev hep 0 değerini döndürecektir. 0 Değeri de unsigned char türünden kuyruklanabilir bir değerdir. Dönen değerin doğrulanması veya ayrımının yapılması programcının sorumluluğudur. Bunu yapmanın en kolay yolu bu işlevi kullanmadan önce kuyruğu; uckuyrukBos() işlevini kullanarak yoklamaktır.

Parametreler

in	*kuyruk*	Uygulamada tanımlanan bir uckuyruk_t yapısına referans.

Döndürdüğü değer

kuyruk NULL ise hep 0, yoksa kuyruğun başındaki veri.

Ayrıca Bakınız

uckuyrukKuyruktanAl()

6.6.3.2 uckuyrukBos()

```
char uckuyrukBos (
            puck_t kuyruk )
```

Bir kuyruğun boş olup olmadığını yoklar.

Parametreler

in	*kuyruk*	Uygulamada tanımlanan bir uckuyruk_t yapısına referans.

Döndürdüğü değer

Kuyruk NULL veya boşsa **1** (true), değilse **0** (false) döndürür.

6.6.3.3 uckuyrukBosalt()

```
void uckuyrukBosalt (
            puck_t kuyruk )
```

Bir kuyruğu tamamen boşaltır (flush).

Bu işlem önce tamponu 0 değeri ile doldurur ve ardından kuyruk yönetim değişkenlerini sıfırlar.

Parametreler

in	*kuyruk*	Uygulamada tanımlanan bir uckuyruk_t yapısına referans.

Ayrıca Bakınız

uckuyrukNBosalt().

6.6.3.4 uckuyrukCokluAl()

```
unsigned char uckuyrukCokluAl (
            puck_t kuyruk,
            unsigned char * hedef,
            const unsigned char nicelik )
```

Bir kuyruktan istenen nicelikte veri alır.

Kuyruğun başındaki verileri kuyruktaki sırasıyla, hedef belleğe belirtilen, nicelikte alır. Kuyruktan alınan veriler kuyruktan çıkarılır. Kuyruktan alınacak veriyi saklayacak kadar yere sahip bir hedef sağlamak programcının sorumluluğudur. Aksi takdirde veriler kaybolabilir veya gereken kapasiteye sahip bir yer sağlanmamışsa verilerin hedefe sığan bölümü alınırken kalanı kaybolabilir. Hedefe aktarma işlemi başarılıysa aktarılan veri niceliği döndürülür. Diğer yandan; kuyruk veya hedef parametrelerinin biri NULL ise veya nicelik 1'den küçükse veya kuyrukta istenen nicelikte veri yoksa 0 değeri döndürülür.

Parametreler

in	*kuyruk*	Uygulamada tanımlanan bir uckuyruk_t yapısına referans.
out	*hedef*	Kuyruktan alınan verilerin saklanmak istediği konuma referans.
in	*nicelik*	Başlangıcı verilen veriden kaç tanesinin kuyruğa alınacağı.

Döndürdüğü değer

Kuyrukta istenen nicelikte veri yoksa **0**, hedefe aktarım başarılıysa aktarılan verilerin niceliği.

6.6.3.5 uckuyrukCokluKuyrukla()

```
unsigned char uckuyrukCokluKuyrukla (
            puck_t kuyruk,
            const unsigned char * kaynak,
            const unsigned char nicelik )
```

Bir kuyruğa belirtilen nicelikte veri ekler.

Veriler kuyruğun en sonuna eklenir ve kuyruklanan verinin niceliği döndürülür. kuyruk veya kaynak parametreleri NULL verilirse veya nicelik 1'den küçükse veya kuyrukta; kuyruklanmak istenen verileri alacak kadar yer yoksa işlem sürdürülmez ve kuyruklamanın başarısız olduğunu ifade eden bir 0 değeri döndürülür.

Parametreler

in,out	*kuyruk*	Uygulamada tanımlanan bir uckuyruk_t yapısına referans.
in	*kaynak*	Veriyi içeren dizideki verinin başlangıç adresine referans.
in	*nicelik*	Başlangıcı verilen veriden kaç tanesinin kuyruğa alınacağı.

Döndürdüğü değer

kuyruk veya kaynak NULL ise veya kuyrukta yeteri kadar boş yer yoksa **0**, kuyruklama başarılıysa **nicelik**.

6.6.3.6 uckuyrukDoldur()

```
unsigned char uckuyrukDoldur (
            puck_t kuyruk,
            const unsigned char deger,
            const unsigned char nicelik )
```

Bir kuyruğu belirtilen sayıda aynı veri ile doldurur.

İşlem başarılıysa kuyruğa doldurulan nicelik döndürülür. kuyruk NULL ise veya kuyrukta doldurulacak nicelikte yer yoksa veya nicelik 1'den küçükse hep 0 değeri döndürülür.

Parametreler

in,out	*kuyruk*	Uygulamada tanımlanan bir uckuyruk_t yapısına referans.
in	*deger*	Kuyruğa doldurulacak unsigned char türünde veri.
in	*nicelik*	Verilen verinin kuyruğa çoğaltılma sayısı.

Döndürdüğü değer

Kuyruk doluysa veya belirtilen nicelikteki verileri alacak kadar boş yer yoksa verileri kuyruğa almaz ve 0, işlem başarılı olursa kuyruklanan verilerin niceliğini döndürür.

6.6.3.7 uckuyrukDolu()

```
char uckuyrukDolu (
            puck_t kuyruk )
```

Bir kuyruğun dolu olup olmadığını yoklar.

Parametreler

in	*kuyruk*	Uygulamada tanımlanan bir uckuyruk_t yapısına referans.

Döndürdüğü değer

Kuyruk NULL veya doluysa **1** (true), değilse **0** (false) döndürür.

6.6.3.8 uckuyrukIlkle()

```
void uckuyrukIlkle (
            const unsigned char kapasite,
            puck_t kuyruk,
            unsigned char * tampon )
```

Bir kuyruğu ilk kullanıma hazırlar.

Kuyruk veya tampon parametrelerinin biri NULL ise işlem sürdürülmez.

Parametreler

in	*kapasite*	Kuyruğun 1 - 255 aralığında kapasitesi, sizeof ile unsigned char türüne cast edilerek verilebilir.
in	*kuyruk*	Uygulamada tanımlanan bir uckuyruk_t yapısına referans.
in	*tampon*	Uygulamada tanımlanan bir unsigned char türünde dizi.

6.6.3.9 uckuyrukKalanKapasite()

```
unsigned char uckuyrukKalanKapasite (
            puck_t kuyruk )
```

Bir kuyruğun kalan kapasitesini verir.

Kuyruğun kapasite niceliğini yani kuyruğun daha kaç öğe alabileceğinin sayısının bilgisini verir.

Parametreler

in	*kuyruk*	Uygulamada tanımlanan bir uckuyruk_t yapısına referans.

Döndürdüğü değer

Kuyruğun kalan kapasitesi, kuyruk NULL ise hep **0**.

6.6.3.10 uckuyrukKuyrugaAktar()

```
char uckuyrukKuyrugaAktar (
            puck_t kaynak,
            puck_t hedef )
```

Kaynak kuyruğun başındaki ögeyi hedef kuyruğa kuyruklar.

Kuyruktan kuyruğa bir öge aktarımı başarılı olursa 1 değeri döndürülür. Kaynak veya hedef kuyrukların referanslarının herhangi birinin NULL olması durumunda işlem sürdürülmez ve 0 değeri döndürülür. Kaynak kuyrukta hiçbir öge yoksa işlem sürdürülmez ve 0 değeri döndürülür. Hedef kuyrukta 1 öge alacak kadar yer yoksa işlem sürdürülmez ve 0 değeri döndürülür.

Parametreler

in	*kaynak*	Kaynak kuyruğa referans.
in,out	*hedef*	Hedef kuyruğa referans.

Döndürdüğü değer

Aktarma başarılıysa **1**, değilse **0**.

Ayrıca Bakınız

uckuyrukKuyrugaCokluAktar(), uckuyrukKuyrugaKopyala().

6.6.3.11 uckuyrukKuyrugaCokluAktar()

```
unsigned char uckuyrukKuyrugaCokluAktar (
            puck_t kaynak,
            puck_t hedef,
            const unsigned char nicelik )
```

Kaynak kuyruktan hedef kuyruğa birden çok öge aktarır.

Kuyruktan kuyruğa aktarım başarılı olursa aktarılan ögelerin sayımı döndürülür. Kaynak veya hedef kuyrukların referanslarının herhangi birinin NULL olması durumunda işlem sürdürülmez ve 0 değeri döndürülür. Kaynak kuyrukta aktarılmak istenen nicelik kadar öge yoksa işlem sürdürülmez ve 0 değeri döndürülür. Hedef kuyrukta aktarılmak istenen nicelik kadar öge yoksa işlem sürdürülmez ve 0 değeri döndürülür. Aktarılmak istenen nicelik değeri 0 verilirse işlem sürdürülmez ve 0 değeri döndürülür.

Parametreler

in	*kaynak*	Kaynak kuyruğa referans.
in,out	*hedef*	Hedef kuyruğa referans.
in	*nicelik*	Aktarılmak istenen öge sayısı.

Döndürdüğü değer

Aktarma başarılıysa aktarılan öge sayımı, değilse **0**.

Ayrıca Bakınız

uckuyrukKuyrugaAktar(), uckuyrukKuyrugaCokluKopyala().

6.6.3.12 uckuyrukKuyrugaCokluKopyala()

```
unsigned char uckuyrukKuyrugaCokluKopyala (
            puck_t kaynak,
            puck_t hedef,
            const unsigned char nicelik )
```

Kaynak kuyruğun başından itibaren nicelik kadar ögeyi hedefe kopyalar.

uckuyrukKuyrugaCokluAktar() İşlevinden farkı kaynak kuyruktan hedef kuyruğa kopyalanan ögelerin kaynak kuyruktan silinmemesidir. Böylece aynı ögeler hem kaynak kuyrukta hem de hedef kuyrukta bulunur. Kopyalama işlemi başarılı olursa hedef kuyruğun sayımı nicelik kadar (kopyalanan ögeler kadar) artarken, ögeler kaynak kuyruktan çıkarılmadığı için kaynak kuyruğun sayımı olduğu gibi kalır ve baştaki öge değişmez. Kuyruktan kuyruğa aktarım başarılı olursa aktarılan ögelerin sayımı döndürülür. Kaynak veya hedef kuyrukların referanslarının herhangi birinin NULL olması durumunda işlem sürdürülmez ve 0 değeri döndürülür. Kaynak kuyrukta aktarılmak istenen nicelik kadar öge yoksa işlem sürdürülmez ve 0 değeri döndürülür. Hedef kuyrukta aktarılmak istenen nicelik kadar öge yoksa işlem sürdürülmez ve 0 değeri döndürülür. Aktarılmak istenen nicelik değeri 0 verilirse işlem sürdürülmez ve 0 değeri döndürülür.

Parametreler

in	*kaynak*	Kaynak kuyruğa referans.
in,out	*hedef*	Hedef kuyruğa referans.
in	*nicelik*	Aktarılmak istenen öge sayısı.

Döndürdüğü değer

Aktarma başarılıysa aktarılan öge sayımı, değilse **0**.

Ayrıca Bakınız

uckuyrukKuyrugaKopyala(), uckuyrukKuyrugaCokluAktar().

6.6.3.13 uckuyrukKuyrugaKopyala()

```
char uckuyrukKuyrugaKopyala (
            puck_t kaynak,
            puck_t hedef )
```

Kaynak kuyruğun başındaki ögeyi hedef kuyruğa kopyalar.

uckuyrukKuyrugaAktar() İşlevinden farkı kaynak kuyruktan hedef kuyruğa kopyalanan ögenin kaynak kuyruktan silinmemesidir. Böylece aynı öge hem kaynak kuyrukta hem de hedef kuyrukta bulunur. Kopyalama işlemi başarılı olursa hedef kuyruğun sayımı bir artarken, öge kaynak kuyruktan çıkarılmadığı için kaynak kuyruğun sayımı olduğu gibi kalır ve baştaki öge değişmez. Kuyruktan kuyruğa bir öge kopyalama başarılı olursa 1 değeri döndürülür. Kaynak veya hedef kuyrukların referanslarının herhangi birinin NULL olması durumunda işlem sürdürülmez ve 0 değeri döndürülür. Kaynak kuyrukta hiçbir öge yoksa işlem sürdürülmez ve 0 değeri döndürülür. Hedef kuyrukta 1 öge alacak kadar yer yoksa işlem sürdürülmez ve 0 değeri döndürülür.

Parametreler

in	*kaynak*	Kaynak kuyruğa referans.
in,out	*hedef*	Hedef kuyruğa referans.

Döndürdüğü değer

Aktarma başarılıysa **1**, değilse **0**.

Ayrıca Bakınız

uckuyrukKuyrugaCokluKopyala(), uckuyrukKuyrugaAktar().

6.6.3.14 uckuyrukKuyrukla()

```
char uckuyrukKuyrukla (
            puck_t kuyruk,
            const unsigned char deger )
```

Bir kuyruğa veri ekler.

Veri kuyruğun en sonuna eklenir ve kuyruklamanın başarılı olduğunu belirten 1 degeri döndürülür. kuyruk parametresi NULL verilirse veya kuyruk doluysa işlem sürdürülmez ve kuyruklamanın başarısız olduğunu ifade eden bir 0 değeri döndürülür.

Parametreler

in,out	*kuyruk*	Uygulamada tanımlanan bir uckuyruk_t yapısına referans.
in	*deger*	Kuyruğa alınacak unsigned char türünde veri.

Döndürdüğü değer

Kuyruk NULL veya doluysa **0**, kuyruklama başarılıysa **1**.

6.6.3.15 uckuyrukKuyruktanAl()

```
unsigned char uckuyrukKuyruktanAl (
            puck_t kuyruk )
```

Bir kuyruğun en başındaki veriyi alır.

Kuyruktan alınan veri kuyruktan çıkarılır. Kuyruğun en başındaki değere onu kuyruktan çıkarmadan bakmak için uckuyrukBastakiOge() kullanılır. kuyruk parametresi NULL verilirse bu işlev hep 0 değerini döndürecektir. 0 Değeri de unsigned char türünden kuyruklanabilir bir değerdir. Dönen değerin doğrulanması veya ayrımının yapılması programcının sorumluluğudur. Bunu yapmanın en kolay yolu bu işlevi kullanmadan önce kuyruğu; uckuyrukBos() işlevini kullanarak yoklamaktır.

Parametreler

in	*kuyruk*	Uygulamada tanımlanan bir uckuyruk_t yapısına referans.

Döndürdüğü değer

kuyruk NULL ise hep 0, yoksa kuyruğun başındaki veri.

Ayrıca Bakınız

uckuyrukBastakiOge(), uckuyrukBos().

6.6.3.16 uckuyrukNBosalt()

```
void uckuyrukNBosalt (
            puck_t kuyruk,
            unsigned char n )
```

Bir kuyruktan n kadar veriyi boşaltır.

Boialtılmak istenen miktar n parametresi ile verilir. Boşaltma işlemi başarılı olursa; en baştaki ögeden itibaren n kadar öge kuyruktan silinir, silinen ögelerin konumları 0 değeri ile doldurulur. Kuyruk parametresi NULL ise veya kuyrukta silinmek istenenden daha az öge varsa veya n değeri 0 verilirse işlem sürdürülmez.

Parametreler

in,out	*kuyruk*	İşlem yapılacak kuyruğa referans.
in	*n*	Boşaltma miktarı.

Ayrıca Bakınız

uckuyrukBosalt().

6.6.3.17 uckuyrukOgeSayimi()

```
unsigned char uckuyrukOgeSayimi (
            puck_t kuyruk )
```

Bir kuyrukta bekleyen öğe niceliğini döndürür.

Parametreler

in	*kuyruk*	Uygulamada tanımlanan bir uckuyruk_t yapısına referans.

Döndürdüğü değer

Kuyrukta bekleyen öğe niceliği, kuyruk NULL ise her zaman **0**.

6.7 MPLAB XC8 PIC18 Portu

Dosyalar

- dosya port.c
- dosya portmacro.h

Fonksiyonlar

- void portKritikBolumGirisi (void)
- void portKritikBolumCikisi (void)

6.7.1 Ayrıntılı tanımlama

Görevci tarafından kullanılacak sistem donanımları burada ilklenir. Herbir portun görevcisi kendi port.c dosyasında gerçeklenmiş olan portGorevciyiBaslat() işlevini çağırarak başlatılır.

6.7.2 Fonksiyon Dokümantasyonu

6.7.2.1 portKritikBolumCikisi()

```
void portKritikBolumCikisi (
             void  )
```

Sistemin kritik bir bölümden çıkmasını sağlar.

Kritik bir işlem biter bitmez bu işlev çağrılmalıdır ki sistemin kesmeye bağlı olay kaynakları hemen etkinleştirilsin ve bu olaylar bir an önce işlensin. Bu yüzden kritik bölüm olabildiğince kısa ve öz olmalıdır.

Ayrıca Bakınız

portKritikBolumGirisi().

6.7.2.2 portKritikBolumGirisi()

```
void portKritikBolumGirisi (
             void  )
```

Sistemin kritik bir bölüme geçmesini sağlar.

Kritik bölüm, genelde atomik yani hiçbir şekilde bölünmemesi gereken bir işlemin hemen öncesinde çağrılmalıdır. Atomik işlemin bitiminde ise bu işlevin eşi olan portKritikBolumCikisi() işlevi çağrılmalıdır. Yoksa kesme bitleri bu işlev tarafından değiştirildiğinden sistem beklendiği gibi çalışmaz.

Uyarı

Bu işlev çağrıldıktan sonra sistemin tüm kesme kaynakları etkisizleştirileceğinden, portKritikBolumCikisi() çağrılana dek hiçbir kesme isteği işlenemeyecektir. Bu yüzden kritik bölüm işlemi olabildiğince kısa ve öz olmalıdır. Yoksa zaman kısıtlı olaylar gibi önemli kesme olayları geç işlenir ve bu olaylara bağlı diğer işlemlerin gecikmesine veya aksamasına neden olur.

Ayrıca Bakınız

portKritikBolumCikisi().

Bölüm 7

Veri Yapıları Dokümantasyonu

7.1 Bayrak Yapı(Struct) Referans

`#include <bayrak.h>`

Veri Alanları

- unsigned char **sinir**

  En çok erişim miktarı.
- unsigned char **kalan**

  Kalan erişim hakkı.

7.1.1 Ayrıntılı tanımlama

Görevlerin veri kaynaklara erişiminin senkronizasyonu için kullanılır.

Bir kaynak veya veri birden fazla koşut (paralel) çalışan görev tarafından kullanıldığında veya değiştirildiğinde, veri bütünlüğünü korumak zor bir iştir. Bayrak (semaphore) API'si bu zor işi yüklenir, görevlerin kaynak veya verilere erişimini senkronize eder.

Bu yapı(struct) için dokümantasyon aşağıdaki dosyadan üretilmiştir:

- /home/Kasa/projeler/gomulu/coklu-gorevleme/gorevci/gorevci/include/bayrak.h

7.2 GorevKontrolBlogu Yapı(Struct) Referans

`#include <gorev.h>`

Veri Alanları

- sn_t **sn**

  Görevin kaldığı yerden devam edebilmesi için sürdürme noktasını tutar.
- unsigned char **kimlik**

  Görevi yönetmek için bir kimlik bilgisi tutar.
- char **durum**

  Görevin çalışma durumu bilgisini tutar.
- is_t **is**

  Görevin işlevinin referansını tutan işlev gösterici (function pointer).

7.2.1 Ayrıntılı tanımlama

Görev Kontrol Bloğu.

Görev Kontrol Bloğu (GKB) görevlerin yönetimi için; göreve ait gerekli bilgileri barındırmak için kullanılır.

Bu yapı(struct) için dokümantasyon aşağıdaki dosyadan üretilmiştir:

- /home/Kasa/projeler/gomulu/coklu-gorevleme/gorevci/gorevci/include/gorev.h

7.3 SureKontrolBlogu Yapı(Struct) Referans

```
#include <gorev.h>
```

Veri Alanları

- unsigned int **baslangic**

  Gecikme istendiği anki sistem tiki sayımını tutar.
- unsigned int kacTik

7.3.1 Ayrıntılı tanımlama

Görevleri geciktirmek için süre kontrol bloğu.

7.3.2 Alan Dokümantasyonu

7.3.2.1 kacTik

```
unsigned int kacTik
```

Geciktirme API'lerine parametre olarak verilen milisaniye türünden süre değerinin sistem tiki türünden karşılığını tutar.

Bu yapı(struct) için dokümantasyon aşağıdaki dosyadan üretilmiştir:

- /home/Kasa/projeler/gomulu/coklu-gorevleme/gorevci/gorevci/include/gorev.h

7.4 UCKuyruk Yapı(Struct) Referans

`#include <uckuyruk.h>`

Veri Alanları

- unsigned char **kapasite**

  1 - 255 arası kapasite. Bu, kuyruğun sonunu bilmek için gereklidir.
- unsigned char sayim
- unsigned char **bas**

  Kuyrukta sıradaki okunmayı / alınmayı bekleyen verinin konumunu tutar.
- unsigned char * tampon

7.4.1 Ayrıntılı tanımlama

UCKuyruk (unsigned char kuyruk) FIFO (İlk giren ilk çıkar) türünde bir kuyruk yapısıdır. Amacı statik olarak tanımlanmış unsigned char türünde bir diziye referans alıp düşük belleğe sahip aygıtlarda basit bir FIFO kuyruğu yönetmektir. Kuyruk, bas ve sayim adında iki değişkenle yönetilir. bas, Kuyruktan alınmayı bekleyen ilk öğenin konumunu tutar.

7.4.2 Alan Dokümantasyonu

7.4.2.1 sayim

`unsigned char sayim`

Kuyruğa yazılan verilerin sayımını tutar. Bir sonraki yazma konumunu hesaplamak için de kullanılır.

7.4.2.2 tampon

`unsigned char* tampon`

Uygulamada oluşturulan unsigned char türünde bir diziye başvuru adresini tutan imci (pointer).

Bu yapı(struct) için dokümantasyon aşağıdaki dosyadan üretilmiştir:

- /home/Kasa/projeler/gomulu/coklu-gorevleme/gorevci/gorevci/include/uckuyruk.h

Bölüm 8

Dosya Dokümantasyonu

8.1 /home/Kasa/projeler/gomulu/coklu-gorevleme/gorevci/gorevci/gorev.c Dosya Referansı

```
#include "gorev.h"
#include "portlar/mplabx/xc8/pic18/port.h"
#include "portmacro.h"
#include "gorevciypl.h"
```

- unsigned int **grvTikSayimini** ()
- void grvTikKesmeIsleyici ()

8.1.1 Ayrıntılı tanımlama

Yazar

İsmail Sahillioğlu (Kozmotronik)

Copyright

MIT Lisansı

8.2 /home/Kasa/projeler/gomulu/coklu-gorevleme/gorevci/gorevci/include/bayrak.h Dosya Referansı

```
#include "gorev.h"
```

Veri Yapıları

- struct Bayrak

Makrolar

- #define grvBAYRAK_ILKLE(b, s)
- #define grvBAYRAK_BEKLE(g, b)
- #define grvBAYRAK_IMLE(b)

Typedef'ler

- typedef Bayrak **bayrak_t**

  Kolaylık sağlamak için Bayrak tür tanımlaması.

8.2.1 Ayrıntılı tanımlama

Yazar

İsmail Sahillioğlu (Kozmotronik)

Copyright

MIT Lisansı

8.3 /home/Kasa/projeler/gomulu/coklu-gorevleme/gorevci/gorevci/include/gorev.h Dosya Referansı

```
#include "sn.h"
#include "portmacro.h"
#include "gorevciypl.h"
```

Veri Yapıları

- struct GorevKontrolBlogu
- struct SureKontrolBlogu

Makrolar

- #define NULL ((void∗) 0)
- #define **grvCALISMA_KIPI** 1

  Görev çalışma kipi tanımlanmamışsa varsayılan normal kip.
- #define **grvTIK_SURESI_uS** 1000u

  Varsayılan tik süresi 1 ms.
- #define **grvTIK_SURESI_MS** (grvTIK_SURESI_uS / 1000u)

  Tik süresinin milisaniye türünden değeri.
- #define grvILKLE(g)
- #define grvBASLA(g)
- #define grvBITIR(g)
- #define grvGECIK_MS(g, s, gecikme)

- #define grvKOSULLU_GECIK_MS(g, s, gecikme, kosul)
- #define grvKOSUL_BEKLE(g, kosul)
- #define grvBU_KOSULDA_BEKLE(g, kosul)
- #define grvSIFIRLA(g)
- #define grvCIK(g)
- #define grvVAZGEC(g)
- #define grvKOSULA_DEK_VAZGEC(g, kosul)

Süre Birimi Dönüştürücüleri

Mikrosaniye <-> tik; milisaniye <-> tik süre dönüşümleri

- #define **grvuS_TIK_CEVIR**(us) (us / grvTIK_SURESI_uS)
  Mikrosaniye süre değerini tik süre değerine dönüştürür.
- #define **grvMS_TIK_CEVIR**(ms) (ms / grvTIK_SURESI_MS)
  Milisaniye süre değerini tik süre değerine dönüştürür.
- #define **grvTIK_uS_CEVIR**(tik) (tik ∗ grvTIK_SURESI_uS)
  Tik süre değerini mikrosaniye süre değerine dönüştürür.
- #define **grvTIK_MS_CEVIR**(tik) (tik ∗ grvTIK_SURESI_MS)
  Tik süre değerini milisaniye süre değerine dönüştürür.

Görev Durum Kodları

Görevlerin işletimdeki durumlarını belirten kodlardır.

- #define **grvBEKLIYOR** 0
  Görev bir olayın gerçekleşmesini bekliyor.
- #define **grvCIKTI** 1
  Görev işlemin herhangi bir noktasında çıktı (çalışmasını sonlandırdı).
- #define **grvBITTI** 2
  Görev işlemi tamamlayarak bitti.
- #define **grvVAZGECTI** 3
  Görev işlemin herhangi bir noktasında gönüllü olarak vazgeçti.

Typedef'ler

- typedef void ∗ **gorevTutucu_t**
  Görevin işlevinin kendisine referansı (handle for function).
- typedef char(∗ **is_t**) (gorevTutucu_t)
  Görevin işlev türü (function pointer type).
- typedef struct GorevKontrolBlogu **gorev_t**
  Kolaylık sağlamak için GorevKontrolBlogu tür tanımı.
- typedef gorev_t ∗ **pgkb_t**
  GKB için referans türü (pointer).
- typedef struct SureKontrolBlogu **sure_t**
  Kolaylık sağlamak için SureKontrolBlogu tür tanımı.

Fonksiyonlar

- unsigned int grvTikSayimi (void)
- void grvTikKesmeIsleyici (void)
- pgkb_t grvOlustur (is_t is)
- void grvGorevciyiBaslat (void)
- pgkb_t grvKimlikIleGorevBlogunuAl (const unsigned char kimlik)
- void grvBaslat (const unsigned char kimlik)
- void grvDurdur (const unsigned char kimlik)

8.3.1 Ayrıntılı tanımlama

Yazar

İsmail Sahillioğlu (Kozmotronik)

Copyright

MIT Lisansı

8.4 /home/Kasa/projeler/gomulu/coklu-gorevleme/gorevci/gorevci/include/sn.h Dosya Referansı

Makrolar

- #define SN_ILKLE(sn)
- #define SN_BASLAT(sn)
- #define SN_KUR(sn)
- #define SN_BITIR(sn)

Typedef'ler

- typedef unsigned int **sn_t**
  Sürdürme Noktası.

8.4.1 Ayrıntılı tanımlama

Yazar

İsmail Sahillioğlu (Kozmotronik)

Copyright

MIT Lisansı

8.4.2 Makro Dokümantasyonu

8.4.2.1 SN_BASLAT

```
#define SN_BASLAT(
              sn )
```

Sürdürme noktasının switch-case yapı ağacını başlatır.

Parametreler

sn	sn_t türünde sürdürme noktası değişkeni.

8.4.2.2 SN_BITIR

```
#define SN_BITIR(
              sn )
```

Sürdürme noktası switch-case yapı ağacını sonlandırır.

Parametreler

sn	sn_t türünde sürdürme noktası değişkeni.

8.4.2.3 SN_ILKLE

```
#define SN_ILKLE(
              sn )
```

Sürdürme noktasının değerini ilkler.

Parametreler

sn	sn_t türünde sürdürme noktası değişkeni.

8.4.2.4 SN_KUR

```
#define SN_KUR(
              sn )
```

Çağrıldığı yere bir sürdürme noktası kurar.

Parametreler

sn	sn_t türünde sürdürme noktası değişkeni.

8.5 /home/Kasa/projeler/gomulu/coklu-gorevleme/gorevci/gorevci/include/uckuyruk.h Dosya Referansı

Veri Yapıları

- struct UCKuyruk

Makrolar

- #define NULL ((void∗) 0)

Typedef'ler

- typedef struct UCKuyruk **uckuyruk_t**

  Kolaylık sağlaması için UCKuyruk tür tanımı.
- typedef uckuyruk_t ∗ **puck_t**

  uckuyruk_t yapısı için başvuru türü (okunaklılığı iyileştirmek için)

Fonksiyonlar

- void uckuyrukIlkle (const unsigned char kapasite, puck_t kuyruk, unsigned char ∗tampon)
- char uckuyrukKuyrukla (puck_t kuyruk, const unsigned char deger)
- unsigned char uckuyrukCokluKuyrukla (puck_t kuyruk, const unsigned char ∗kaynak, const unsigned char nicelik)
- unsigned char uckuyrukBastakiOge (puck_t kuyruk)
- unsigned char uckuyrukKuyruktanAl (puck_t kuyruk)
- unsigned char uckuyrukCokluAl (puck_t kuyruk, unsigned char ∗hedef, const unsigned char nicelik)
- char uckuyrukKuyrugaAktar (puck_t kaynak, puck_t hedef)
- unsigned char uckuyrukKuyrugaCokluAktar (puck_t kaynak, puck_t hedef, const unsigned char nicelik)
- char uckuyrukKuyrugaKopyala (puck_t kaynak, puck_t hedef)
- unsigned char uckuyrukKuyrugaCokluKopyala (puck_t kaynak, puck_t hedef, const unsigned char nicelik)
- void uckuyrukBosalt (puck_t kuyruk)
- void uckuyrukNBosalt (puck_t kuyruk, unsigned char n)
- unsigned char uckuyrukDoldur (puck_t kuyruk, const unsigned char deger, const unsigned char nicelik)
- unsigned char uckuyrukOgeSayimi (puck_t kuyruk)
- char uckuyrukDolu (puck_t kuyruk)
- unsigned char uckuyrukKalanKapasite (puck_t kuyruk)
- char uckuyrukBos (puck_t kuyruk)

8.5.1 Ayrıntılı tanımlama

Yazar

İsmail Sahillioğlu (Kozmotronik)

Copyright

MIT Lisansı

8.6 /home/Kasa/projeler/gomulu/coklu-gorevleme/gorevci/gorevci/portlar/mplabx/xc8/pic18/port.c Dosya Referansı

```
#include <xc.h>
#include "gorev.h"
```

Fonksiyonlar

- void portKritikBolumGirisi ()
- void portKritikBolumCikisi ()

8.6.1 Ayrıntılı tanımlama

Yazar

İsmail Sahillioğlu (Kozmotronik)

Copyright

MIT Lisansı

8.7 /home/Kasa/projeler/gomulu/coklu-gorevleme/gorevci/gorevci/portlar/mplabx/xc8/pic18/port.h Dosya Referansı

```
#include "gorevciypl.h"
#include "gorev.h"
```

Fonksiyonlar

- void portKritikBolumGirisi (void)
- void portKritikBolumCikisi (void)

8.7.1 Ayrıntılı tanımlama

Yazar

İsmail Sahillioğlu (Kozmotronik)

Copyright

MIT Lisansı

8.8 /home/Kasa/projeler/gomulu/coklu-gorevleme/gorevci/gorevci/portlar/mplabx/xc8/pic18/portmacro.h Dosya Referansı

```
#include <xc.h>
```

Makrolar

- #define **portKESME_ONCELIK_KONTROL_BITI** 0x7
  Kesme öncelik kontrol biti numarası.
- #define **portKESME_ONCELIK_KONTROL_MASK** 0x80
  Kesme öncelik kontrol biti için maskeleme.
- #define **portGLOBAL_KESME_YETKI_BITI** 0x7
  Küresel kesme yetki biti numarası.
- #define **portGLOBAL_KESME_YETKI_MASK** 0x80
  Küresel kesme yetki biti için maskeleme.
- #define **portCEVRESEL_KESME_YETKI_BITI** 0x6
  Çevresel kesme yetki biti numarası.
- #define **portCEVRESEL_KESME_YETKI_MASK** 0x40
  Çevresel kesme yetki biti için maskeleme.
- #define **portGLOBAL_YUKSEK_ONCELIK_KESME_YETKI_BITI** 0x7
  Yükesk öncelikli kesme yetki biti numarası.
- #define **portGLOBAL_YUKSEK_ONCELIK_KESME_YETKI_MASK** 0x80
  Yüksek öncelikli kesme yetki biti için maskeleme.
- #define **portGLOBAL_DUSUK_ONCELIK_KESME_YETKI_BITI** 0x6
  Düşük öncelikli kesme yetki biti numarası.
- #define **portGLOBAL_DUSUK_ONCELIK_KESME_YETKI_MASK** 0x40
  Düşük öncelikli kesme yetki biti için maskeleme.
- #define **portGLOBAL_KESME_KAPAT**() INTCON &= ~(1 << portGLOBAL_KESME_YETKI_BITI)
  Küresel kesme yetki bitini etkisizleştirir.
- #define **portGLOBAL_KESME_AC**() INTCON |= 1 << portGLOBAL_KESME_YETKI_BITI
  Küresel kesme yetki bitini etkinleştirir.
- #define **portCEVRESEL_KESME_KAPAT**() INTCON &= ~(1 << portCEVRESEL_KESME_YETKI_BITI)
  Çevresel kesme yetki bitini etkisizleştirir.
- #define **portCEVRESEL_KESME_AC**() INTCON |= 1 << portCEVRESEL_KESME_YETKI_BITI
  Çevresel kesme yetki bitini etkinleştirir.
- #define portNOP()

8.8.1 Ayrıntılı tanımlama

Yazar

İsmail Sahillioğlu (Kozmotronik)

Copyright

MIT Lisansı

Mimari portuna özel tanımlamalar.

Bu dosyadaki tanımlamalar Görevcinin çalıştığı donanım için düzgünce yapılandırılmasını sağlar. Bu yüzden bu ayarlar değiştirilmemelidir.

8.8.2 Makro Dokümantasyonu

8.8.2.1 portNOP

```
#define portNOP( )
```

Hedef porta özel NOP gerçeklemesi.

8.9 /home/Kasa/projeler/gomulu/coklu-gorevleme/gorevci/gorevci/uckuyruk.c Dosya Referansı

```
#include "uckuyruk.h"
```

Fonksiyonlar

- void uckuyrukIlkle (const unsigned char kapasite, puck_t kuyruk, unsigned char ∗const tampon)
- char uckuyrukKuyrukla (puck_t kuyruk, const unsigned char deger)
- unsigned char uckuyrukCokluKuyrukla (puck_t kuyruk, const unsigned char ∗kaynak, const unsigned char nicelik)
- unsigned char uckuyrukBastakiOge (puck_t kuyruk)
- unsigned char uckuyrukKuyruktanAl (puck_t kuyruk)
- unsigned char uckuyrukCokluAl (puck_t kuyruk, unsigned char ∗hedef, const unsigned char nicelik)
- char uckuyrukKuyrugaAktar (puck_t kaynak, puck_t hedef)
- unsigned char uckuyrukKuyrugaCokluAktar (puck_t kaynak, puck_t hedef, const unsigned char nicelik)
- char uckuyrukKuyrugaKopyala (puck_t kaynak, puck_t hedef)
- unsigned char uckuyrukKuyrugaCokluKopyala (puck_t kaynak, puck_t hedef, const unsigned char nicelik)
- void uckuyrukBosalt (puck_t kuyruk)
- void uckuyrukNBosalt (puck_t kuyruk, unsigned char n)
- unsigned char uckuyrukDoldur (puck_t kuyruk, const unsigned char deger, const unsigned char nicelik)
- unsigned char uckuyrukOgeSayimi (puck_t kuyruk)
- char uckuyrukDolu (puck_t kuyruk)
- char uckuyrukBos (puck_t kuyruk)
- unsigned char uckuyrukKalanKapasite (puck_t kuyruk)

8.9.1 Ayrıntılı tanımlama

Yazar

İsmail Sahillioğlu (Kozmotronik)

Copyright

MIT Lisansı

Dizin