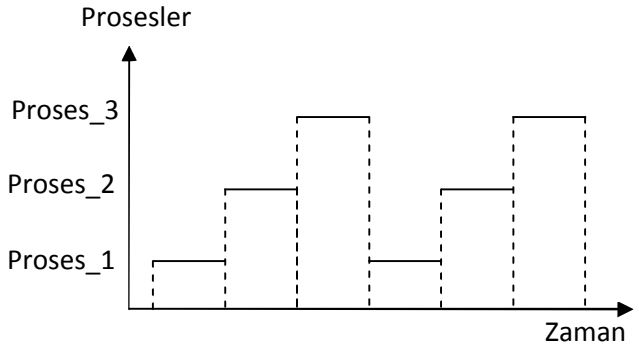
3. PROSES YÖNETİMİ (PROCESS MANAGEMENT)

Bir bilgisayar sisteminde çalışan tüm yazılımlar, proses olarak anılır. Genel olarak bir sistemde, kullanıcı ya da işletim sisteminin prosesleri vardır. Bir proses, sistem çağrılarını kullanarak başka bir proses oluşturabilir ya da diğer proseslerle etkileşim kurabilir.

Proses daha teknik bir tabirle, çalışır durumda olan programa verilen bir isimdir. CPU tarafından işletilmeyen bir program pasif bir yapıya sahip olup komutlar dizisinden ibarettir. Bu komutlar işletime alınmak suretiyle işlevlerini yerine getirebilirler.

Tüm modern bilgisayarlar aynı anda birkaç işi yapabilirler; kullanıcı programı çalışıyorken diskten okuma, bir text dokümanı ekranda gösterme veya çıktı işlemlerini bir arada gerçekleştirebilir. Çoklu programlamayı destekleyen sistemlerde CPU, bir programdan diğer bir programa belli zaman dilimlerinde anahtarlanır. Aslında CPU aynı anda bir programı çalıştırabilmesine rağmen, prosesler arasında hızlı anahtarlandığından dolayı bu prosesleri paralel çalıştırdığı izlenimini verir.



CPU'nun proseslere anahtarlanması

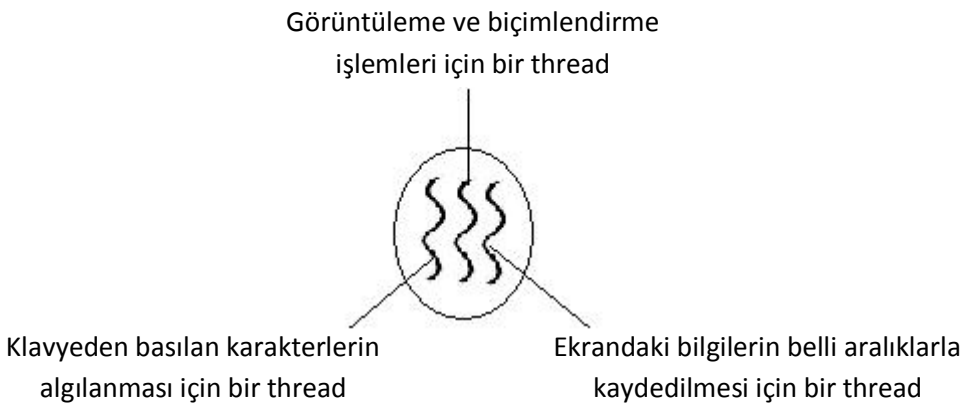
Grafikten, belli bir zaman aralığında tüm proseslere çalışma imkanının verildiği görülmektedir.

İşletim sistemine ait görevler (spooling gibi), toplu işler (batch job), kullanıcı programları birer prosestir. Prosesler CPU, bellek, dosya ya da giriş/çıkış işlemleri gerektirebilirler. Bu kaynaklar ya proses oluşturulurken ya da çalıştırılırken proseslere tahsis edilirler.

3.1. Klasik ve Modern Prosesler

Teknolojik gelişmeler sonucunda proses kavramı iki farklı manaya gelmektedir. Klasik proses; von Neumann mimarisi bilgisayarlarda işletilmekte olan programa ait bir tabirdir. 80'li 90'lı yıllarda uygulama programları için yeterli olmuştur. Daha sonraları modern proses ve thread (iplik) kavramı ortaya çıkmıştır. Modern prosesler bir stüdyo gibi, thread'ler ise müzik yapmak için stüdyoyu kullanan müzisyenler gibi düşünülebilir. Klasik proseslerde, her bir müzisyen, kendi stüdyosuna sahipken, modern proseslerde müzisyenler (thread) ortak bir stüdyoyu paylaşabilirler. Programcılar, programlarının değişik kısımlarını bir thread kümesi olarak tek bir klasik proses çatısı altında geliştirebilirler. Klasik prosesler, tek bir thread'e sahip modern prosesler olarak düşünülebilir.

Bir bilgisayar sisteminde her bir prosesin kendi adres uzayı vardır. Prosesler bir arada çalışabilirler ancak bazı özel mekanizmalar (shared memory gibi) dışında adres uzaylarını paylaşmazlar. Bir proses çatısı altındaki thread'ler ise aynı adres uzayını, global değişkenleri, açık durumda olan dosyaları paylaşabilirler. Her thread'in kendine has durumları (hazır, bekler,…), program sayacı (PC), stack bölgesi, yerel değişkenleri ve saklayıcıları vardır. Bir prosesin birden fazla yürüteceği işlem varsa, işlemleri bölmek yani thread'lere paylaştırmak performansı arttırabilir. Bu durumda bir thread bloke olduğunda diğer thread'ler çalışabilecektir. Thread'lerin kendi kaynakları olmadığı için oluşturulmaları ve sonlandırılmaları, proseslere göre daha kolaydır. Özellikle thread'lerin bir kısmı işlemciye yönelik diğer bir kısmı da I/O işlemlerine yönelik çalıştırılabilirse daha yüksek performans elde edilir. Örneğin bir kelime işlemci programı düşünüldüğünde, bir thread basılan karakterlerin algılanması için, diğer thread basılan karakterlerin ekranda gösterilmesi ve biçimsel değişiklikler için, başka bir thread de belli aralıklarla içeriğin kaydedilmesi için bir arada kullanılırsa, tek bir prosese göre daha yüksek performans gözlenecektir.



Kelime işlemci programında kullanılabilecek thread'ler

Çok işlemcili sistemlerde, her bir işlemciye bir thread atamak suretiyle gerçek paralellik sağlanabilir. Tek işlemcili sistemlerde ise, bir prosese adanan sürede, prosesin sahip olduğu thread'ler zaman paylaşımlı olarak çalışırlar dolayısıyla gerçek paralellik sağlanamaz.

90'lı yılların öncesinde çoklu programlama, senkronizasyon ve kilitlenme gibi kavramlar klasik prosesler üzerinde yorumlanmıştır. Bugün ise tüm bu başlıklar genelleştirilerek, proses ve thread kavramına uygun hale getirilmiştir. Dolayısıyla işletim sistemini klasik proses kavramı üzerine öğrenmek ve daha sonra modern proses ve thread kavramına göre uydurmak daha kolay olacaktır.

Günümüz işletim sistemlerinden olan FreeBSD Unix, modern prosesleri destekler şekilde tasarlanmamasına rağmen, Linux ve Solaris gibi işletim sistemleri klasik proses kavramı üzerine inşa edilmiş ve bazı eklentilerle (kütüphane rutinleriyle) modern proses ve thread kavramlarını destekler hale getirilmiştir. Mach ve Windows işletim sistemleri modern proses ve thread kavramı üzerine kurulmuştur. Unix ailesinin klasik prosesleri, Windows ailesinin de modern prosesleri desteklemek üzere tasarlandığı söylenebilir.

3.2. Sistem Çağrıları

Bilgisayar ilk açıldığı zaman işletim sistemi ana belleğe yüklenir ve çalışmaya başlar. İşletim sistemi bir programı çalıştırmak üzere seçtiğinde o programın bulunduğu ana bellek bölgesine dallanır ve programı işletir. Belirli bir zaman diliminden sonra işletim sistemi tekrar CPU'nun kontrolünü kazanır ve kendi kodunu çalıştırır. Ve bu durum sürekli olarak devam eder. Ancak kullanıcı programları ayrıcalıklı komutları çalıştıramazlar; kullanıcı adına işletim sistemi servisleri tarafından yapılır. Uygulama programları, sistem çağrılarını kullanarak bu servisleri kullanırlar. Sistem çağrıları, proses yönetimi, aygıt yönetimi, bellek yönetimi ve dosya yönetimi kapsamında işletim sistemi tarafından ele alınır. Örneğin proses yönetiminde; Unix'te fork(), Windows'ta ise CreateProcess() sistem çağrıları proses oluşturmak için kullanılır. Benzer şekilde Linux'ta pthread_create() ve Windows'ta CreateThread() thread oluşturmak için kullanılır. Proses yöneticisi kullanıcı komutlarının ve gerektiğinde ayrıcalıklı komutların işletiminden sorumludur.

Özellikle Unix'in birçok türevinin çıkmasıyla birlikte farklı sistem çağrı arabirimleri kullanılmaya başlanmıştır. Bu farklılıkları gidermek için POSIX.1 standardı geliştirilmiştir. Unix tabanlı sistemler (Linux, OpenBSD, FreeBSD) bu çağrı arabirimini baz alırken Windows ailesinin üyeleri Win32 API'yi kullanır.

Proses ve threadlerin oluşturulması ve sonlandırılması, proseslere kaynak tahsisi, proseslerin giriş/çıkış işlemleri için aygıt yöneticisiyle ve belleğe yüklenmesi için de bellek yöneticisiyle işbirliği yapmak proses yöneticisinin sorumluluğundadır.

3.3 Kullanıcı ve Supervisor Mod

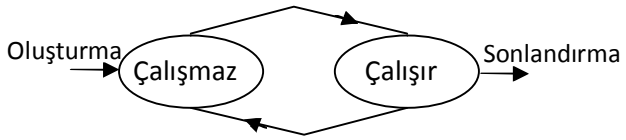
İşletim sistemi, kaynak paylaşımı ve korumanın sağlanabilmesi için giriş/çıkış komutları gibi ayrıcalıklı komutları kendi bünyesinde işler. Böyle olmasaydı, kullanıcı örneğin disk sürücünün herhangi bir kısmını istediği gibi okuyup yazabilirdi. Ayrıcalıklı komutlar, işletim sistemi tarafından CPU'nun supervisor (veya kernel) modunda işletilir. Bir kullanıcı programındaki tüm komutlar, kullanıcı modu komutlarıdır ve donanım üzerinde direkt olarak icra edilir. Kullanıcı programı, örneğin proses oluşturmak için fork() komutunu kullandığında donanım tarafından direkt olarak değil CPU'nun supervisor modunda işletim sistemi fonksiyonu olarak işletilir. Bu komutlar fonksiyon çağrıları olarak bilinir. Linux'ta birkaç yüz, Windows'ta ise birkaç bin sistem çağrısı vardır. Windows'ta sistem çağrılarının daha fazla olmasının sebeplerinden biri masaüstü pencere sisteminin işletim sisteminin bir parçası olmasıdır.

3.4. Proses Kontrol Bloğu ve Proses Durum Diyagramları

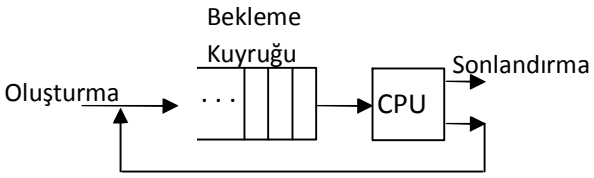
Bir prosesi sadece program kodlarından ibaret olduğunu düşünmek yeterli olmaz. Çoklu programlamada, değişik zaman dilimlerinde farklı prosesler çalıştığından proseslere ait bilgilerin tutulduğu proses kontrol blokları (process control block) vardır. Bu bloklar, prosesin ana bellekteki yeri, hafıza limitleri, proses aktivitesi ('hazır', 'çalışır' ya da 'bekler' gibi), program sayacının değeri, işlemcinin kaydedicileri (EAX, EBX,…), prosesin ID numarası, prosesin önceliği, tahsis edilen kaynaklar, açık durumdaki dosyalar gibi bilgileri içerirler.

3.4.1. İki durumlu proses modeli

Herhangi bir zaman diliminde bir proses CPU tarafından ya işletiliyordur ya da işletilmiyordur. Yani proses ya çalışır ya da çalışmaz durumdadır. İşletim sistemi tarafından bir proses oluşturulduğunda çalışır durumda değildir. Bir kesme vuku bulduğunda ya da zamanlayıcı işlevi gerçekleştiğinde yeni bir proses çalıştırılmak üzere seçilecektir ve çalışır durumdaki proses çalışmaz durumuna geçecektir. Çalışır durumda olmayan prosesler daha sonra çalıştırılmak üzere kuyrukta tutulurlar. Kuyruktan da ilk gelen ilk çıkar (FIFO) mantığıyla veya proseslere öncelik değeri atamak suretiyle proses seçilebilir.



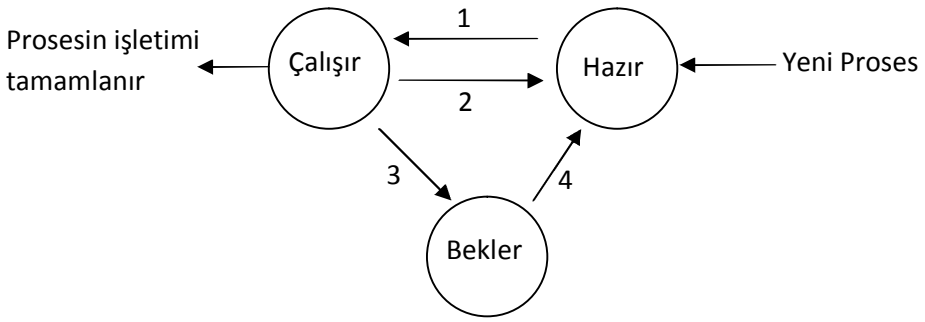
Durum geçişleri



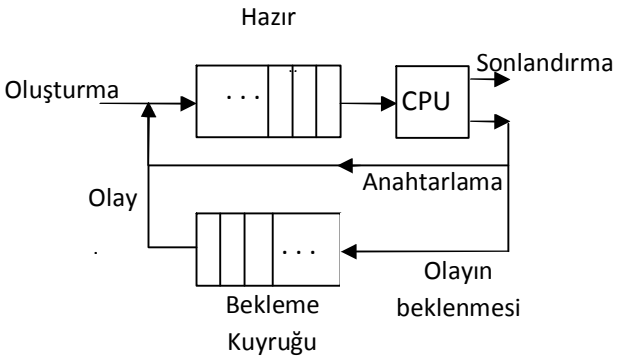
Kuyruk diyagramı

3.4.2. Üç durumlu proses modeli

Eğer tüm prosesler çalışmaya hazır durumundaysa, kuyruk yapısı iki durumlu modeldeki gibi düşünülebilir. Fakat giriş/çıkış işleminin sonlanmasını bekleyen bir proses olduğunda kuyruktan sıra ile prosesleri işletime sokmak mümkün olmayacaktır. İki durumlu modeldeki çalışmaz durumu, hazır ve bekler durumu olarak genişletilebilir.



3 durumlu modelin durum geçişleri



Üç durumlu modelin kuyruk diyagramı

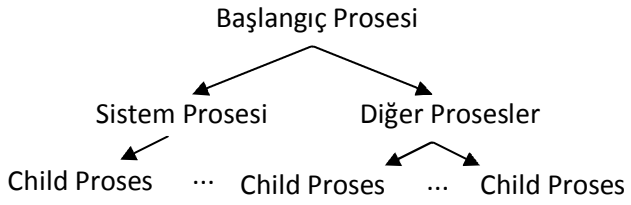
Proses yönetici oluşturduğu prosesi ilk olarak hazır kuyruğuna koyar, daha sonra çalıştırmak üzere bir prosesi seçeceği zaman da yine bu kuyruktan seçim yapar. Çalışan prosesin zaman içerisinde ya işletimi sonlandırılır ya da duruma bağlı olarak hazır veya bekler kuyruğuna konulur. Örneğin proses bir giriş/çıkış işlemine başladığı düşünülürse veya bir olayın gerçekleşmesini bekliyorsa bekler kuyruğuna konur. Prosese ait olay gerçekleştiğinde (veya I/O işlemi tamamlandığında) bekler durumdaki proses hazır durumunu alır. Bir olay vuku bulduğunda (I/O, semafor,…) işletim sistemi bekleme kuyruğunu tarayacak, bu olayı bekleyen prosesleri tespit edecek hazır kuyruğuna aktaracaktır. Kuyrukta yüzlerce proses olduğu düşünüldüğünde her bir olay için bir kuyruk tahsisi yapmak daha verimli bir kullanım olacaktır. Böylece olay vuku bulduğunda ilgili kuyruktaki tüm prosesler hazır kuyruğuna aktarılacaktır.

3.5. Parent (anne) ve Child (çocuk) Proses Kavramı

Bir proses tarafından oluşturulan proses child proses, child prosesi üreten proses de parent proses olarak isimlendirilir. Proses hiyerarşisinde parent proses çok sayıda child prosese sahip olabilirken child proses sadece bir parent prosese aittir.

İşletim sisteminin kendisi ya da kullanıcı, proses oluşturabilir. Örneğin kullanıcı bir dosyayı yazdırmak istediğinde uygulama programı adına işletim sistemi bu işletimi yönetecek bir proses oluşturabilir. Ya da kullanıcı prosesi veri üretirken, bu proses tarafından oluşturulacak diğer bir proses eşzamanlı olarak verileri uygun bir formata çevirebilir. Bu durum yapısal programlamada oldukça faydalıdır.

Genellikle işletim sistemi yüklendiğinde bir başlangıç prosesi otomatik olarak oluşturulur ve bu proses tüm proses hiyerarşisinin kökü olur. Oluşturulan her proses de bu köke eklenir.



Proses hiyerarşisi