

ÖNSÖZ

Tez çalışması sırasında bana her türlü desteği sağlayan, konu üzerindeki birikimini tezime aktaran, tezimin formata uygunluğunu sağlayan; tezin oluşmasını sağlayan değerli danışman hocam Doç. Dr. Coşkun Sönmez'e teşekkürlerimi sunarım.

Tez öncesinde ve tez aşamasında sağlamış olduğu maddi ve manevi destekten dolayı Eliar Elektronik A.Ş. çalışanlarına ve özellikle yöneticimiz Mehmet Taygun ve proje arkadaşlarım Hasan Hüseyin Yılmaz, Okan Vural ve Yunus Altunbıçak' a teşekkürlerimi sunarım.

Tezin şekillenmesinde ve öncesinde, yüksek lisans eğitimim esnasında sağlamış olduğu vazgeçilmez desteği sebebi ile arkadaşım Araş. Gör. Sanem Sarıel'e teşekkürlerimi sunarım.

Şu ana dek almış olduğum eğitim sürecinde tarafıma her türlü katkıda bulunan ve bilgisayar mühendisliğini bana sevdiren ve hayatıma kazıyan İzmir Çınarlı Teknik Lisesi Bilgisayar Bölümü hocalarıma, sonrasında İstanbul Teknik Üniversitesi Kontrol ve Bilgisayar Mühendisliği Bölümü hocalarıma ve nihayetinde İstanbul Teknik Üniversitesi Bilgisayar Yüksek Mühendisliği hocalarıma teşekkürlerimi sunarım.

Bu tezin üzerinde doğrudan veya dolaylı yönden katkıları bulunan, eğitim ve akademi camiasındaki yazar ve araştırmacılara, kontrolör ve denetçilere, basım ve yayıncılara, arkadaş ve dostlarıma, yazılımcılara teşekkürlerimi sunar başarılarının devamını dilerim...

Aralık 2002

İbrahim Niyazi ÜLGÜR

İÇİNDEKİLER

KISALTMALAR	v
TABLO LİSTESİ	vi
ŞEKİL LİSTESİ	vii
ÖZET	viii
SUMMARY	xi
1. GİRİŞ	1
2. GÖMÜLÜ SİSTEMLER	5
2.1 Donanım	5
2.1.1 Motorola Kartı	7
2.1.2 Intrinsyc Kartı	7
2.1.2.1 Özellikleri	8
2.1.2.2 Bellek Haritası	9
2.2 Yazılım	9
2.2.1 Linux	9
2.2.2 Derleyici & Bağlayıcı	10
2.3 Sonuçlar ve Tartışma	11
3. INTRINSYC LINUX : I-LINUX	12
3.1 Çekirdek (Kernel)	13
3.2 Çekirdeğin başlaması	14
3.2.1 Kuyruklar (Queues)	18
3.2.2 Zamanlayıcı (Timer)	18
3.2.3 Seviyeler	19
3.2.4 Kesme (Interrupt)	19
3.3 Modüller	20
3.4 Prosesler	21
3.4.1 Proses durumları	21
3.4.2 Proses zamanlamaları	22
3.5 Dosya sistemleri	22
3.6 Ağ özellikleri	24
3.7 Bazı komutlar	24
4. IPC – PROSESLER ARASI İLETİŞİM	25
4.1 Paylaşılmış bellek (Shared Memory)	25
4.1.1 Semaforlar	26

4.2 Sinyaller (Signals)	27
4.3 Soketler (Sockets)	28
5. TASARIM	30
5.1 Tasarım Modelleme Dili : UML	30
5.2 Sistem aktörleri	31
5.3 Blok Diyagram	32
5.4 IO Yapılandırması	34
5.4.1 PLC IO yapılandırması	34
5.4.2 Tasarım IO yapılandırması	35
5.4.3 Önerilen PLC iletişimi Tasarım	36
5.4.4 IO Kullanılması	38
5.5 Algoritmalar (Prosesler)	38
5.6 Komutlar	41
5.7 Program (Yığın işlem)	42
5.8 Yorumlayıcı	45
5.9 Organizatör (Çalıştırıcı)	46
5.9.1 Linux zamanlayıcısının kullanılması	47
5.9.2 Kilitlenmeler ve çözümler	48
5.9.3 Yarış durumları ve çözümler	49
5.9.4 Meşgul döngüler (Busy waiting)	49
5.10 Sonuçlar ve Tartışma	50
6. SİSTEM YAZILIM MİMARİSİ VE MODULLER	51
6.1 IO Yönetim Modülü	53
6.1.1 PLC	53
6.1.2 Operatör Terminali	55
6.2 Algoritmalar	57
6.3 Komut Editör Modülü	61
6.4 Program Editör Modülü	62
6.5 Yorumlayıcı Modül	63
6.6 Organizatör Modül	64
6.7 Sonuçlar ve Tartışma	65
6.7.1 Gerçek zaman kriterleri	65
7. SONUÇLAR	67
8. TARTIŞMA	68
KAYNAKLAR	71
ÖZGEÇMİŞ	73

KISALTMALAR

UML : Unified Modelling Language
XML : Extended Mark-up Language
GUI : Graphical User Interface
PLC : Programmable Logic Controller
CPU : Central Processing Unit
DSP : Digital Signal Processor
IPC : Inter Process Communication
PID : Proportional + Integral + Derivative Control
KB : Kilo Byte
MB : Mega Byte
TCP : Transmission Control Protocol
FTP : File Transfer Protocol
PPP : Point to Point Protocol
SMTP : Simple Mail Transfer Protocol
RAM : Random Access Memory
ROM : Read Only Memory
EEPROM : Electrically Erasable Programmable ROM
PCB : Printed Circuit Board
MMU : Memory Management Unit
RISC : Reduced Instruction Set Computing
POSIX : Portable Operating Systems for Interface Extension
DMA : Data Memory Area
SQL : Structural Query Language

TABLO LİSTESİ

Sayfa No

Tablo 2.1. Intrinsyc kart bellek haritası	9
---	---

ŞEKİL LİSTESİ

Sayfa No

Şekil 2.1 Kibrit kutusu PC : Küçük giyilebilir bir platform	5
Şekil 2.2 Intrinsyc kartın önden ve arkadan görüntüsü.....	8
Şekil 3.1 Linux işletim sistemi blok diyagramı	12
Şekil 3.2 Linux çekirdeğinin başlama prosedürü.....	14
Şekil 3.3 init() yordamının içeriği.....	17
Şekil 4.1 Paylaşılmış bellek kullanımı	25
Şekil 4.2 Sunucu ve İstemci açılarından soket haberleşmesi.....	29
Şekil 5.1 Sistemde tanımlı kullanıcılar	31
Şekil 5.2 Sunrise, UML bileşen diyagramı	33
Şekil 5.3 Terminaldeki verilerin fiziksel giriş çıkış birimlerine aktarılması	35
Şekil 5.4 İki parçalı tasarım	36
Şekil 5.5 2 Durumlu bir iletişim makinesi	37
Şekil 5.6 4 durumlu bir iletişim makinesi – OMRON PLC iletişim protokolü	37
Şekil 5.7 IO API’si use case diyagramı	38
Şekil 5.8 Linux ve Sunrise’ın benzerlikleri	39
Şekil 5.9 Algoritma akış diyagramı	39
Şekil 5.10 Algoritma bileşenleri	40
Şekil 5.11 Komutlar	42
Şekil 5.12 Yığın işlem standart adım pozisyonları	43
Şekil 5.13 Program durumları.....	44
Şekil 5.14 Program başlatma prosedürü	45
Şekil 5.15 Organizatör kullanım şart diyagramı	46
Şekil 5.16 Algoritma durum diyagramı	47
Şekil 5.17 Algoritmaların sonlandırılması	48
Şekil 5.18 Algoritma döngüsünde sürekli beklemenin önlenmesi.....	50
Şekil 6.1 İki parçalı tasarım	52
Şekil 6.2 Algoritma giriş ve çıkışları	58
Şekil 6.3 Organizatör ve algoritmaların ilişkisi	64

PROSES YÖNETİCİSİ BİR GÖMÜLÜ SİSTEM TASARIMI : SUNRISE

ÖZET

Gömülü sistemler gün geçtikçe hayatımızda daha büyük yer kaplamaktadır. Günümüz teknolojisindeki gelişmeler daha yetenekli ve güçlü sistemleri oluşturmakta ve bu sistemler birçok amaca hizmet etmektedir. 1996 yılında var olan ve masa üstüne sığmayan sistemler günümüzde avuç içine sığmakta ve fiyatları da gittikçe düşmektedir. Bugün elektronik olarak kontrol edilen ve gömülü bir sistem içermeyen cihaz kalmamıştır.

Ülkemizde bu tip cihazlar kullanılmakla birlikte henüz bu tip cihazların tasarlanmasında ve üretilmesinde yeterli bir noktaya gelinememiştir. Bu tez ile gömülü sistemler ve yığın kontrol hakkında ve tasarımları konusunda birçok araştırmacı ve tasarımcıya yardımcı olmak ve yön göstermek hedef alınmıştır.

Gömülü sistemler her sayısal sistemde olduğu gibi yazılım ve donanım bütünlüğünden oluşur. Bu tezde daha çok yazılım bölümü ile ilgilenilmiştir.

Gömülü sistemlerde gerek yazılım gerekse donanım olarak birçok çözüm mevcuttur. Bütün bu çözümlerin kendine has özellikleri vardır. Dolayısı ile, seçilen bir çözümün iyi veya kötü olması hakkında yorum yapılamaz. Bu konuda sadece amaca uygunluğu hangi oranda karşıladığı önem arz etmektedir. Bu tez çalışması esnasında birçok olanak irdelenmiş ve bu tez en doğru bulunan çözümler üzerine yazılmıştır.

Gömülü sistemlerde amaç kontrol üzerine şekillenmektedir. Proses kontrol ise daha çok endüstriyel cihazlarda, daha zor ortamlarda, daha zor kontroller anlamına gelmektedir. Örneğin, tekstil sektöründeki bir boyama işlemi, boyama prosesi esnasında, çok titizlik gerektirir. Bu esnada kullanılan kimyasallar içerisinde enzimler bulunmaktadır. Enzimler, organik bileşiklerdir. Bu enzimler 53 °C sıcaklıkta, prosese tabi tutulmalıdır. Diğer sıcaklıklarda ise enzimler zarar görmekte ve oldukça değerli olan kumaşlar yanlış boyandıklarından dolayı zarar görmektedir.

Gömülü sistemlerde çözümlerin çokluğu nedeni ile çözüme yaklaşımlar da fark göstermektedir. Bu tezde gömülü bir sistemde, sistemin yazılım tarafında, proses kontrole yönelik genel bir kontrol sistemi oluşturulacak ve bu sistemin yumuşak gerçek zamanlı prosesleri ne şekilde kontrol edebileceği üzerinde yorumlar yapılacaktır.

Gömülü sistemde, işletim sistemi kullanılması uygun görülmüş ve işletim sistemi olarak Linux seçilmiştir. Tez içerisinde neden Linux seçildiği, tartışmaları ve sonuçları ile birlikte sunulacaktır.

Günümüzdeki tüm yaygın işletim sistemleri kendi içerisinde proses yönetim sistemi (IPC) içermektedir. Bahsi geçen bu proses yönetim sistemi, farklı işletim sistemlerinde, aynı görevleri yapmaktadır. Proses yönetim sistemleri, prosesler arasındaki iletişimi, bir dizi yöntemi kullanarak gerçekleştirir. Bu tezde, Linux işletim sistemindeki IPC (Prosesler arası iletişim sistemi) kullanılmıştır. Böylece işletim sisteminin sağladığı tüm olanaklar kullanılmıştır.

İşletim sistemi ve proses yönetim sistemi kararı üzerine artık problemi tanımlamak ve bu problemi çözecek olan tasarımı yapmak kalmaktadır. Bu tezde bu tasarım incelemektedir. Tezin asıl konusu olan tasarım, proses kontrolüne yönelik olacaktır. Endüstriyel proses kontrolünden bahsettiğimizde, en yaygın olan cihazlar PLC'lerdir.

PLC'ler değişik şekillerde kullanılabilirler. Genel olarak, 2 kullanım biçimi söz konusudur. İlk kullanım biçiminde, PLC programlanabilen bir cihaz değil, sanki bir IO yapısı imiş gibi görülür. Bu kullanım, PLC için çok ilkel bir kullanım şeklidir. Bu durumdaki kullanım, "aptal IO" olarak isimlendirilir. Diğer kullanımda ise, PLC, olması gerektiği gibi, üzerinde bir takım kontrol şartlarını ve/veya programlarını da içerebilir. Bu tezde PLC, mantık kurulacak olan sistemde bulunacağından, aptal IO olarak kullanılacaktır.

PLC, proses kontrol tanımında var olan ve gömülü sisteme entegre edilen mantığı gerçeğe dönüştürür. Bu işlemi yaparken genellikle pnömatik kontrol kullanılır. Bunun sebebi, pnömatik kontrolün yönetilebilir olması ve endüstriyel malzemelerin ihtiyaç duyduğu güce sahip olmasıdır (130 °C'lik buhar vanasının kontrolünü yapmak konunun ne kadar ciddi olabileceğini göstermektedir...).

Bu tezin asıl konusu, proses mantığının, gömülü cihaza ne şekilde aktarılacağıdır. Bu amaç, kullanıcının tanımlayacağı yığın (batch) işlemi, program şeklinde gömülü cihazda koşturmak sayesinde elde edilecektir.

Yığın (batch) işlem, sıralı olarak koşturulan bir dizi işlem bütünüdür. Her bir işlem, adım olarak isimlendirilir. Böyle bir adım kümesinde, sıralı olduğu kadar paralel adımlar da bulunabilir. Böylece bir anda daha çok işlem yapılarak proses süreleri kısaltılabilir. Tez içerisinde anlatılacağı gibi, bu konuda standartlar bulunmaktadır ve kendi başına geniş bir konudur...

Bir program oluşturulabilmesi için kullanıcıya birtakım araçlar sunmak gereklidir. Bu araçlar, nahif kullanıcının anlayabileceği ve tercihen konfigüre edebileceği şekilde olmalıdır. Bu tezde savunulan çözüm gerçekleştiğinde, kullanıcı, programını komutlar kullanarak yazacaktır.

Komutlar, anlaşıldığı gibi yığın işlemin adımlarıdır. Aynı zamanda proses mantığını içeren birimlerdir. Bu tez içerisinde bu mantığa algoritma denilecektir. Algoritmalara bir örnek olarak, kontrol sistemlerinde çok sık kullanılan PID algoritmasını verebiliriz. PID algoritması, bir kazanın sıcaklığını kontrol etmek için kullanılabilir. Bu algoritma, komut olarak düşünüldüğünde, kullanıcıdan bir takım parametreleri alacaktır. Bu parametreler, en basit hali ile P, I, D katsayıları olabilir.

Sistemdeki önemli hususlardan birisi, belki de en önemlisi, sistemin modüler tasarlanma amacıdır. Modüler olmaktan kasıt, istenildiğinde bir algoritmanın kolay

bir şekilde sisteme eklenebilmesi veya çıkarılabilmesidir. Bu durumda ortaya çıkan proses kontrol sistemi, deęişik endüstriyel prosesleri kontrol edebilme yeteneğine sahip olacaktır.

Bu tezdeki proses kontrol sistemi sadece kavramsal olarak tasarlanacaktır yani tasarım ile alakalı olarak program verilmeyecektir. Tezi inceleyen, dilerse istedięi programlama dilini kullanarak kendi programını yazabilir.

SUNRISE, AN EMBEDDED SYSTEM FOR BATCH CONTROLLING, DESIGNED ON LINUX USING IPC

SUMMARY

Embedded systems are becoming popular day by day. The recent technologies could achieve more powerful and talented devices and these devices could manage many tasks easily. The 1996's technologies on desktops are now in palm devices while becoming cheaper. There is no such device that has an electronic control facility without having an embedded system.

In Turkey, there are some embedded devices but unfortunately the design and manufacturing is not enough. One of the main purposes of this thesis is to give readers some information about design and help the reader.

Every embedded system consists software and hardware parts. In this thesis our main scope will be software...

There are too many solutions about software and hardware in embedded systems. Every solution has its own advantages and disadvantages. Based on that no one could talk about which solution would be better than the other. The main problem is that whether the solution could manage the problem or not. In the thesis some solutions will be discussed and some of them will be selected as the best solution.

The main focus in embedded systems is to control the system. Process control means more harshy environments, strict time constraints, industrial devices etc... For example, in textile industry, a dying process needs strict constraints. There is some organic materials in these processes. These organic materials need 53 °C to complete the process. If the temperature changes roughly, these organic materials might be unusable. The result is than, wrong dyed fabrics.

The results might be different because there are too many solutions. In these thesis, a solution which is in the software side and focused on process control will have been found. And the solution will be discussed over the real time considerations and how to control the process.

In our embedded system, an operating system was wanted to be used and Linux was choosed. In the thesis, it will be discussed why Linux was choosed and the results will be represented.

Nowadays, in the operating systems sides, there are some methods to manage the Interprocess Communication (IPC). Some standarts are established on this area. (POSIX). In the thesis, Linux's IPC will be used and the methods will be given.

After the decisions on the Linux and IPC system, the problem will be discussed and the analysis will be done.

The best known devices in industry are the PLCs. There are some different usages for PLCs. Two of them are common. First, the PLC will not be programmed, rather than it will be used for just IO purposes. The usage will be called “dummy IO”. On the other hand, in second, the PLCs can be used with programming it. In the thesis, the first method will be chosen because there will be a controller which contains the process cases and logic.

The main idea in this thesis is to embed the process logic in the embedded device. The aim can be achieved with running a batch program into the embedded device.

The batch process is a program which contains a command array. Each command is called step and parallel steps might be constructed to shorten the program time. There are some standards on this theme and will be mentioned later.

The commands are the steps of the batch process. The commands contain the process logic. We call this logic as algorithm. For example, the PID algorithm is the best known algorithm in control science. PID algorithm can be used to adjust the temperature, to accelerate a motor or to load water. If we consider the PID algorithm as a command, we must supply some variables to algorithm. Simply the P, I and D coefficients must be given.

The most important thing in the design is to modularise the system. That means, the modules can be added to or removed from the system easily. With this feature, the system can be used in different industrial platforms.

In this thesis, the system will be designed conceptually. There will be no code given. The researcher can code his/her own system, with his/her favorite programming language.

1. GİRİŞ

Günümüzde birçok elektronik cihaz, hayatımıza bir şekilde yardımcı olmaktadır. Bu elektronik cihazların artık büyük bir çoğunluğu sayısal devreler ile kontrol edilmektedir. Bu cihazlar hayatımızda öylesine benimsenmişlerdir ki artık üzerinde, özel bir yorum bile yapılmamaktadır. Örneğin günümüzde kullandığımız araçların lastiklerinde dahi sayısal devreler bulunmakta ve lastik, yol ile ilgili bilgileri, araçta bulunan şoför terminaline göndermektedir. Böylece yolcular daha güvenli bir şekilde sürüşünü gerçekleştirirken, lastikteki teknolojiye habersizdir. Benzer örnekleri artırmak mümkündür.

Gömülü Sistemler

Sayısal kontrol söz konusu olduğunda, mikroişlemciler ve bu işlemcilerin oluşturduğu sistemler düşünülmelidir. Değişik tasarımlar için değişik cihazlar üretilmiştir. Sıcak ortamlar için özel ısılara dayanan işlemciler, ses işleme için DSP ismi verilen ses ve görüntü işlemeye özel tasarlanmış işlemciler vb. bunlardandır.

Bu sistemler, cihaz bütün olarak görüldüğünde fark edilmezler, fakat onlar orada bulunurlar ve tüm cihazı kontrol altında tutarlar. Bu sistemler Gömülü Sistemler olarak adlandırılırlar.

Bu tezde, bir gömülü sistem üzerinde, bir yığın işlemcisi tasarımı söz konusudur. Dolayısı ile 2. bölümde öncelikli olarak gömülü sistemler tanıtılacaktır. Mevcut bir takım sistemler ve bu sistemlere ait bilgiler verilecektir.

İşletim Sistemi

Sayısal kontrol cihazlarının gün geçtikçe gelişmesi nedeni ile artık, gömülü sistemler için işletim sistemleri de söz konusu olmaktadır. İşletim sistemleri, 60 KB boyutundan MB'larca boyutlara kadar değişik büyüklüklere sahiptir. Bu kadar çözüm içerisinde, işletim sistemi amaca göre seçilebilir.

İşletim sistemleri, donanımdan bağımsız bir platform sunmaktadır. Bu nedenle tekrar kullanılabilen kodlar yazmak isteyen tasarımcılar, işletim sistemlerini göz önünde bulundurmalıdır.

İşletim sistemleri kendi faydalarını da birlikte getirmektedir. Bu faydaları şu şekilde sıralayabiliriz : ağ erişimi ve ağ programlama ortamını sunması, dosya sistemi sunması, proses yönetimi sunması vb.

Bu bağlamda, tez aşamasında kullanılan işletim sistemi olan I-Linux ve özelliklerini 3. bölümde incelemekteyiz.

Prosesler Arası İletişim (IPC)

İşletim sisteminin çok görevli bir işletim sistemi olması, aynı zamanda o işletim sisteminde proses kavramını da birlikte getirmektedir. Proseslerin ötesinde görev (task), yordam (thread) gibi başka paralel işleme yöntemleri de bulunmaktadır. Fakat ismi ne olursa olsun, önemli olan işletim sisteminin paralel işlem yapıp yapmadığı olacaktır.

Tez için kullandığımız işletim sistemi, Linux türevi olduğundan dolayı proseslerden bahsetmek doğru olacaktır. İşletim sistemi, birden fazla prosesi aynı anda çalıştırabildiğine göre, bu prosesler hakkında iki durum ortaya çıkabilir : İlk durumda prosesler birbirlerinden tamamen bağımsızdır. Gerçekten de işletim sistemi içerisinde, örneğin farklı protokollere servis veren prosesler birbirlerinden tamamen bağımsızdır. FTP protokolü ile SMTP protokolü için yazılmış olan prosesler birbirlerinden bağımsız olacaklardır. İkinci durumda ise prosesler arasında bir iletişim söz konusu olacaktır. Bu durum bizim konumuzda da geçerlidir. Bu gereklilik göz önünde bulundurularak, çok görevli işletim sistemlerine, prosesler arası iletişim yetenekleri eklenmiştir. 4. Bölümde anlatacağımız konu, Linux işletim sisteminde var olan prosesler arası iletişim (IPC) özellikleridir.

Tasarım

Tezin asıl konusu, Linux işletim sistemi üzerinde ve Linux'un özelliklerini de kullanarak bir yığın işlemci sistemi tasarlamaktır. Bu nedenle önceden verilen bilgiler doğrultusunda tasarıma ulaşılabilecektir. Tasarım kavramsal olarak verilecektir.

Günümüz proje analiz ve tasarım araçlarından olan UML ile alakalı bilgiler 5. bölüm başında sunulacak, ardından yığın işlemciye ait tüm kavramlar ayrıntılı bir biçimde incelenecektir.

Mimari ve Modüller

Tasarım, belirli ilkeler doğrultusunda yapılacaktır. Tasarım ilkeleri şunlardır : Kolay anlaşılabilirlik, kolay üretilebilirlik, kolay test edilebilirlik, kolay sürdürülebilirlik ve tekrar kullanılabilirlik. Bu nedenle tasarımı modüler olarak yapmak gereklidir. Bir önceki konuda sistem modüllerine ilişkin sınırlar belirlenmiş olacaktır. Sistem modüllerine ait ayrıntılar 6. bölüm olan “mimari ve modüller” konusunda daha derinlemesine incelenecektir.

Sonuçlar

Teze ait tüm bilgiler verildikten sonra, genel anlamda, yapılmış olan araştırmaya bakılacak ve sonuçları yorumsuz olarak verilecektir. Böylece diğer araştırmacıların, beklentileri ile tezin sunduklarını daha iyi analiz etmelerine olanak sağlanacaktır.

Teze ait sonuçlar 7. bölümde sunulacaktır.

Tartışma

Teknoloji ve bilgidaki yenilikler durmamaktadır. Hatta teknolojiye gelişme birçok durumda kendini geliştirenleri bile şaşırtabilmektedir. Teknolojik bir ürün çıktığında, teknolojisi geçmiş olabilmektedir.

Her araştırmada olduğu gibi, tez sonunda bu araştırmaya genel anlamda bir kez daha bakılacak ve eksikliklere değinilecektir. Gerek zaman gerekse başka nedenlerin yokluğundan ötürü yapılamayan ve gelecekte yapılabilecek olan değişiklikler yorumlanacaktır. Tüm bunlar 8. bölümdeki “Tartışma” başlığı altında yapılacaktır.

Bu tez üzerinde yer alan bilgilerin kaynakları, tezin sonunda yer alan “Kaynaklar” başlığı altında verilecektir.

Tez yazımında verilemeyecek derecedeki uzun açıklamalar, tezin en sonunda ekler bölümünde sunulacaktır.

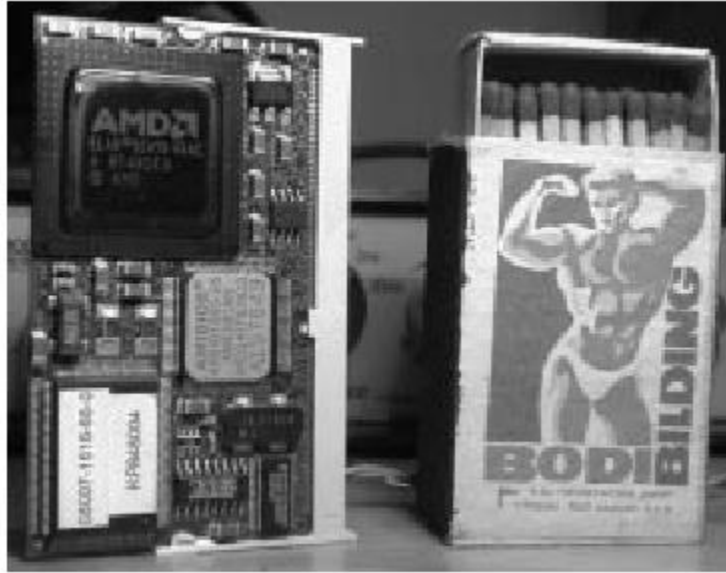
Tezi yapan arařtırmacıya ait bilgiler, daha ileri arařtırma yapmak isteyen arařtırmacılara bilgi sunmak amacı ile yine tezin son bölümünde verilecektir.

2. GÖMÜLÜ SİSTEMLER

Gömülü sistemler iki ana başlık altında incelenmelidir. Bu başlıklar, donanım ve yazılım olacaktır. Bu bölümde gömülü sistemler karşılaştırmalı olarak incelenecektir.

2.1 Donanım

Günümüzde gömülü sistem donanımı ile alakalı olarak birçok çalışma yapılmıştır ve yapılmaktadır. Bunun yanı sıra birçok ticari ürün mevcuttur. Bazıları [1], kibrit kutusu büyüklüğündedir.



Şekil 2.1 Kibrit kutusu PC : Küçük giyilebilir bir platform

Bu ürünlerde işletim sistemleri dahi mevcuttur ve masaüstü bilgisayarların yaptıkları işlerin büyük bir bölümünü de yapabilirler. Bu tip ürünlerde, ilk olarak paketlenme sorunlarını aşmak gereklidir. VLSI ve paketlenme teknolojisindeki yeni gelişmeler ile gömülü cihaz boyutları gittikçe küçülmüştür. Ayrıca artık 4 katlı PCB'ler standart olmuştur [1].

Bu tip ürünlerde yazılım tarafını da hesaba katmak gerekir. Bu ürünlerdeki işletim sistemi ihtiyaçlarından dolayı, bir uyarlanmış işletim sisteminin çalışması gereklidir. Bu ürünlerde kullanılabilecek yazılımın, büyük bir çoğunluğu hazırdır. Hatta, bu nedenle yazılımın bu tip cihazlara gömülmesi eskisine oranla %30 oranında daha kolaydır [1].

Gömülü bir sistemde donanıma ilişkin bir takım kriterler mevcuttur ve bu konuda yapılmış olan araştırmalarda [1,2,3] ortak bir takım özellikler baskın çıkmaktadır :

- CPU : Gömülü sistemin karakterini belirler. Diğer tüm özellikler CPU seçilmesi ile şekillenebilecektir. Gömülü sistemlerde kullanılan CPU'lar öylesine artmıştır ki genel bir klasmanın yapılması söz konusu olmamaktadır. Genellikle x86 serisi kullanılmaktadır [1]. Aslen x86 çıkış amacı olarak, daha genel amaçlı tasarlanmıştı. Fakat zamanla gömülü sistemlere uygun olarak bir takım özellikleri de x86 işlemcisi içerisine gömülmüştür. ISA kontrolcüsü, seri portlar, paralel port, görüntü arabirimi bunlardan bazılarıdır [1]. Alternatif CPU'lar arasında en güçlü olanı ARM işlemcisidir. Bir çok uygulama, özellikle gömülü sistemlerde, ARM üzerinde tasarlanır olmuştur [2,3].
- FLASH : Bir gömülü sistemin en baskın ikinci ögesidir. Veri depolama görevlerinde kullanılır. Masaüstü bilgisayarlardaki sabit disk ögesinin yerini, gömülü cihazlarda flash'lar almaktadır. Fakat bazı uygulamalarda [1], çok küçük boyutlarda sabit diskler kullanılabilmektedir. Aşırı ısı ve titreşim içeren ortamlarda, küçük de olsa hareketli parça içerdiklerinden sabit diskler uygun olmamaktadır. Bunun farkında olan araştırmacılar tasarımlarını artık, Compact Flash ismi verilen ve aslen sabit diskin aynı arabirimine sahip olan, Flash içeren cihazlar kullanarak yapmaktadır [1]. Günümüz Flash cihaz boyutları, 8-16 MB'lar ile ifade edilir iken günümüz gömülü işletim sistemleri 1 MB altındaki boyutlara sığabilmektedir. Bu durum, veri depolama alanları oldukça yüksek hacimli gömülü cihazların oluşturulabilmesini sağlar.

- RAM : Gömülü sistemin performansını doğrudan etkiler. RAM ihtiyacı, o gömülü sistemdeki işletim sistemi ile doğrudan alakalıdır. Gömülü cihazlarda genellikle 16 MB [1,2] yeterli olmaktadır.
- DİĞER : Seri portlar, Paralel portlar, IO pinleri bu tip özelliklerdendir.
- Görüntü sürücüsü : Gömülü sistemlerde, görüntü işlemci genellikle gömülü işlemci (CPU) içinde bulunur [1,2,3].
- Ağ sürücüsü : Gömülü sistemlerin iletişim yeteneklerini artırır. Günümüzde nerede ise her gömülü sistemde [1,2,3], bu tip sürücüler bulunmaktadır.

2.1.1 Motorola Kartı

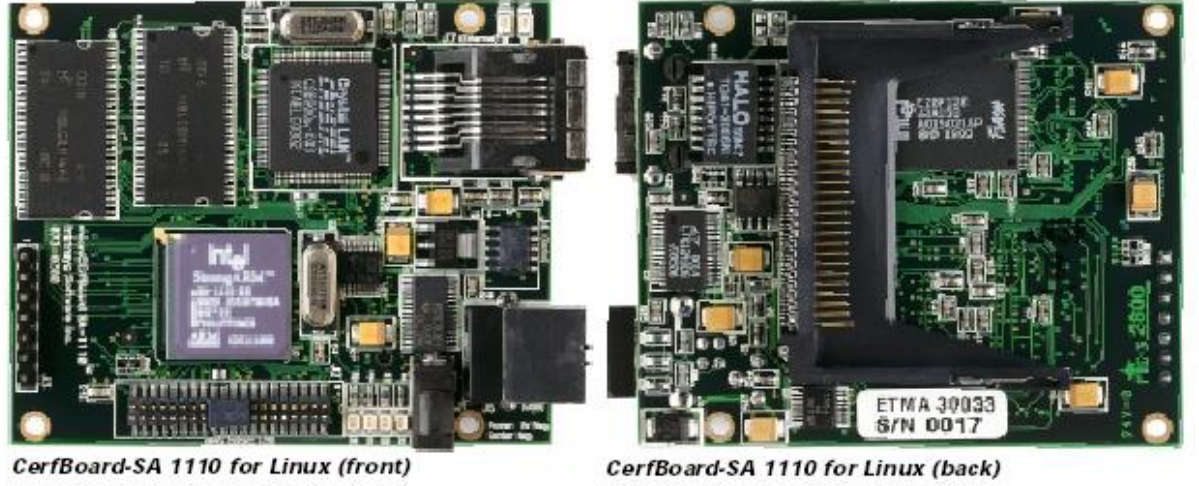
Motorola, gömülü sistemlerde var olan işlemcileri ve çözümleri ile oldukça başarılı bir üreticidir. (www.motorola.com). Motorola'nın gömülü sistemler için üretmiş olduğu özel işlemciler [4] kaynağından incelenebilir.

Tez aşamasında, uygulama safhasında, Motorola 68EZ628 ile yapılmış olan bir kart, pratik olarak incelenmiştir. Özet olarak bu kart, 33 Mhz MC68VZ328 işlemcisi, 2 MB flash, 8 MB RAM, 21 genel amaçlı IO pini, görüntü sürücüsü, 10 Mbit ethernet kontrolcüsü, RS-232 seri port, SPI seri kanal birimlerinden oluşmaktadır.

2.1.2 Intrinsyc Kartı

Intrinsyc [5], SA-1110 CPU kullanarak yapmış olduğu ürüne, CerfCube ismini vermiştir. Bu ürün gömülü cihazlara çok uygun bir örnek teşkil etmektedir. Bu cihazın boyutları 5.7cm x 6,9cm şeklindedir.

Bu üründe kullanılan Strong ARM işlemci çekirdeği (RISC), x86 işlemciye göre çok daha az güç tüketir ve gömülü sistemler için daha uygundur [1]. Gerçekten de ARM işlemcinin, gömülü sistemlerde kullanılması için daha birçok neden mevcuttur.



Şekil 2.2 Intrinsyc kartın önden ve arkadan görüntüsü.

2.1.2.1 Özellikleri

Ürünün temel özellikleri aşağıda sıralanmışlardır :

- SA-1110 işlemci @ 192 MHz
- 16 MB flash
- 32 MB 100 MHZ RAM
- LCD kontrolör
- Flash disk desteği Tip I / II / CF / CF+
- 16 tane genel amaçlı IO pini
- 3 x RS232 port
- USB port
- 10 Base T Ethernet
- 5V ta 400 mA akım çekiyor.

Bu özellikleri ile birlikte, boyutları göz önüne alındığında, gömülü bir sistemin ne derece güçlü bir yapıya ulaştığı anlaşılabilir.

2.1.2.2 Bellek Haritası

Sistemin bellek haritası, sistem programcıları için önem arz eder. Ayrıca tasarımcılar ve donanımcılar için de önemlidir. Bu amaçla aşağıda Intrinsyc kart için bellek haritasını verilmiştir :

Tablo 2.1 Intrinsyc kart bellek haritası

	Block 128 k each		Item		Address Range (hex)	
	Linux	Windows CE	Linux	Windows CE	Linux	Windows CE
Flash	0		Bootloader (I-Boot)		0000 0000	
					0001 FFFF	
	1-2		Bootloader Reserved		0002 0000	
					0005 FFFF	
					0006 0000	
	3-10	3-106	Linux Kernel	Windows CE Image	0015 FFFF	00D5 FFFF
	11-127	107-127	JFFS2 filesystem	IPSM Storage	0016 0000	00D6 0000
Unused					00FF FFFF	
					0100 0000	
RAM					BFFF FFFF	
					C000 0000	
					C1FF FFFF	

2.2 Yazılım

Günümüz gömülü sistemlerinde, işletim sistemleri olmaz ise olmaz koşullardandır. Bu konuda birçok işletim sistemi alternatifi bulunmaktadır. pSOS, vxWorks, EPOC32, Qnx, Linux [3] bunlardan bazılarıdır. Bunların yanında Windows türevi ticari ürünler de bulunmaktadır. Windows CE, Windows XP embedded, Windows NT embedded vb.

2.2.1 Linux

Linux işletim sistemi, son zamanlarda, gömülü sistemler için oldukça popüler olmuştur [7]. Burada Linux'un konfigüre edilebilir bir işletim sistemi olması (çekirdek ve modüller ileride bahsedilecektir), açık kaynak kodu, ücretsiz olması,

çekirdeğinin ufak olması [7], yaygın olması için önem arz eden kriterlerdendir. Bunun ötesinde sağlamış olduğu gömülü sistem geliştirme ortamı da çok uygundur.

Linux, masaüstü bilgisayarlarından, gömülü sistemlere dek birçok sistemde kullanılmaktadır. Bunun nedeni yine çekirdeğin konfigüre edilebilir olmasıdır [7]. Bu aralıktaki bir alana yayılmış olan bu işletim sisteminin değişik türevleri çıkmış ve kaynak kodu açık olduğu için, birçok kişi tarafından kendi amaçları doğrultusunda özelleştirilmiştir.

Linux'un bir türevi, Motorola kart üzerinde bulunan ve bu karta uyarlanmış halidir. Bu sürümün ismi "ucLinux" dur. uCLinux ile alakalı olarak bir takım akademik yayınlar bulunmaktadır [6]. Bu türevin en önemli özelliği, hafıza yönetim sistemi (MMU) içermemesidir. Böylece Linux işletim sisteminin kod olarak boyutu daha da küçülmektedir. Fakat dezavantaj olarak, özellikle işletim sistemindeki bellek güvenlik sorunu ortaya çıkmaktadır.

Linux'un Intrinsyc karta uyarlanmış haline, I-Linux ismi verilmiştir. Bu türev, masaüstü sistemlerinde kullanılan türeve çok benzemektedir. I-Linux ile alakalı olarak daha detaylı bilgi önümüzdeki konularda verilecektir.

2.2.2 Derleyici & Bağlayıcı

Linux işletim sistemi, aslen C ile yazılmıştır [8],7. GNU C derleyicisi C, C++ ve objective C ile yazılmış olan programları derleyebilir. Tüm Linux versiyonlarında GNU C derleyicisi bulunur. Zira çekirdeğin kendisi de bu derleyici ile derlenmektedir. Linux sürümlerinde GNU C derleyicisi "gcc" ismini alır [8],7.

Gcc derleyicisi aynı zamanda bir geçiş-derleyicisidir. (cross-compiler). Yani "gcc" kaynak dosyalarını, uyarlandığı platforma göre derler. Gcc derleyicisi, ANSI standartlarını baz olarak kabul eder ve bunun ötesinde bir takım eklentilere sahiptir [8],22. Eğer yazılmış kodlar, eklentilere sahip değil ise, yani kaynak kod ANSI C standartlarında yazılmış ise bu durumda bu kod, her türlü platform için kolayca derlenebilir. Yani yazılım taşınabilir olur.

Şu ana dek örnek olarak verilen platformlar :

- Kibrit kutusu büyüklüğündeki PC : Burada Intel 486 işlemci kullanıldığından, masaüstü bilgisayarlar ile aynı derleyici kullanılır.
- Motorola kart : Burada “gcc”nin, bu karta uygun versiyonu kullanılır.
- ARM Intrinsyc kart : Burada, Motorola’ya benzer şekilde, ARM işlemci için uyarlanmış derleyici kullanılır. Bu derleyicinin ismi PC’lerde çalışma ortamını karıştırmamak için arm-linux-gcc şeklinde değiştirilmiştir.

Derleme bir prodestir. Bu proses 4 aşamalı olarak gerçekleşir : Önışlem (preprocessing), derleme (compilation), birleřtirme (assembly) ve baęlama (linking) [8],8. Tüm bu işlemler, “gcc” tarafından gerçekleştirilir.

Bir proje söz konusu olduğunda, birden çok dosyanın projeye dahil olması kaçınılmazdır. Bu durumda, her bir dosyanın ayrı derlenmesi gerekir [8],12. Linux’ta bu amaç doğrultusunda “gcc” kullanılabileceęi gibi daha uygun olması açısından, “GNU Make” kullanılır [8],12,31. “make” komutu ile “makefile” ismi verilen dosyalardaki kriterlere göre projeler derlenmektedir. “Make” ile birlikte projelerin daha hızlı derlenmesi, “make” deęiřen dosyaları derlemektedir, mümkündür [8],34.

2.3 Sonuçlar ve Tartışma

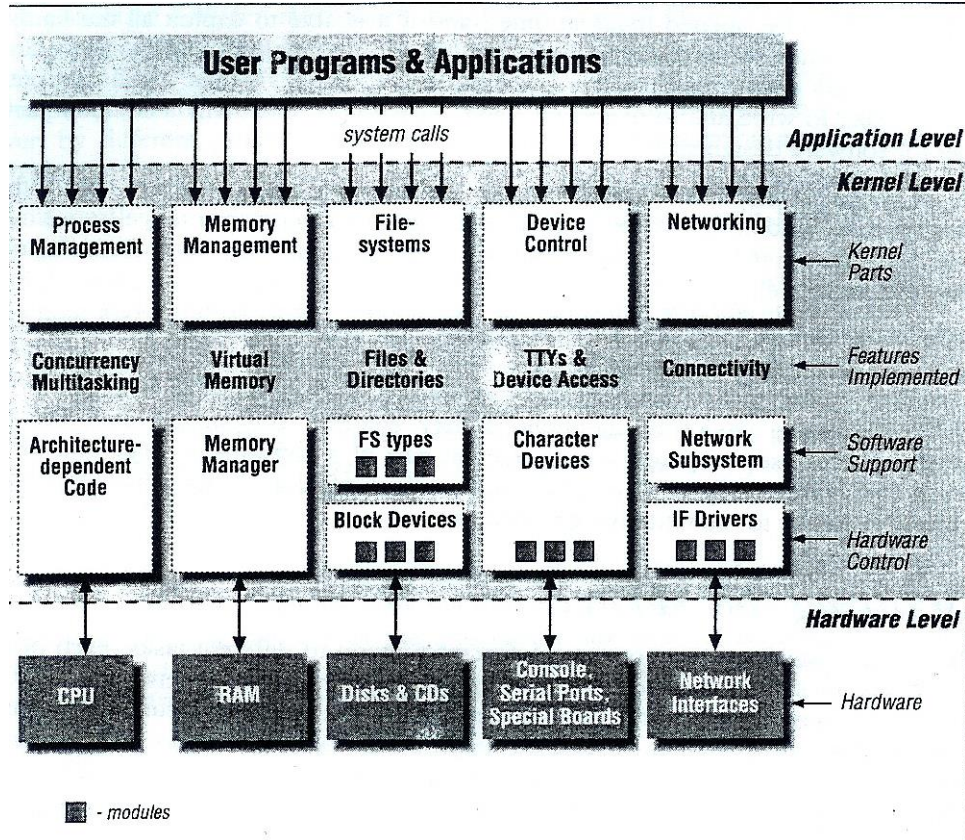
Bu bölümde gömülü sistemlerin, günümüzde geldikleri noktalar ortaya konulmuş ve bu ürünler ile neler yapılabileceęi anlatılmıştır. Burada anlatılan gömülü sistemler dışında birçok sistem mevcuttur.

Bu bölümde verilen örnekler, tezin amaçladığı yığın işlemciyi yapabilecek ve amaca uyabilecek sistemler olarak örnek gösterilmiştir.

Örnek gösterilen sistemler üzerinde ne şekilde bir platform bulunduğu ve bu platformları kullanarak sistemlerin nasıl gerçekleştirilebileceęi anlatılmıştır. Bu bağlamda derleme, geçiş derlemesi , bağlama ve proje yönetimi ile alakalı bilgiler sunulmuştur.

3. INTRINSYC LINUX : I-LINUX

I-Linux, Linux'tan türetme bir işletim sistemidir. Linux işletim sistemi kaynak kodu açıktır. Bu nedenle çeşitli üreticiler, Linux'u, kendi yapmış oldukları platformlara uygun olarak şekillendirmektedirler. Bu şekillendirme, platform bağımlı işlemlerin işletim sistemine tanıtılması şeklinde olacaktır. Örneğin kesmeler platforma uygun bir şekilde ayarlanmalı ve kesme rutinleri ile Linux çekirdeği arasındaki bağlantılar kurulmalıdır. Böylece Linux çekirdeği, o platform üzerinde koşabilecek duruma gelir. Aşağıdaki şekilde donanıma özgü bölümün, Linux'un diğer bölümlerinden ne şekilde ayrıldığı görülebilmektedir.



Şekil 3.1 Linux işletim sistemi blok diyagramı

Bu tezin yazım aşamasındaki en son I-Linux sürümü 4.4 dür [9] I-Linux işletim sistemini kullanabilmek için ilk önce I-Linux dosya sisteminin oluşturulması gereklidir. Ardından Linux çekirdeğinin derlenmesi gerekecektir. Bunun üzerine artık dosya sistemi ve Linux çekirdeğin cihaza gömülmesi mümkün olacaktır [9]. Benzer işlemler diğer platformlar için de söz konusu olacaktır.

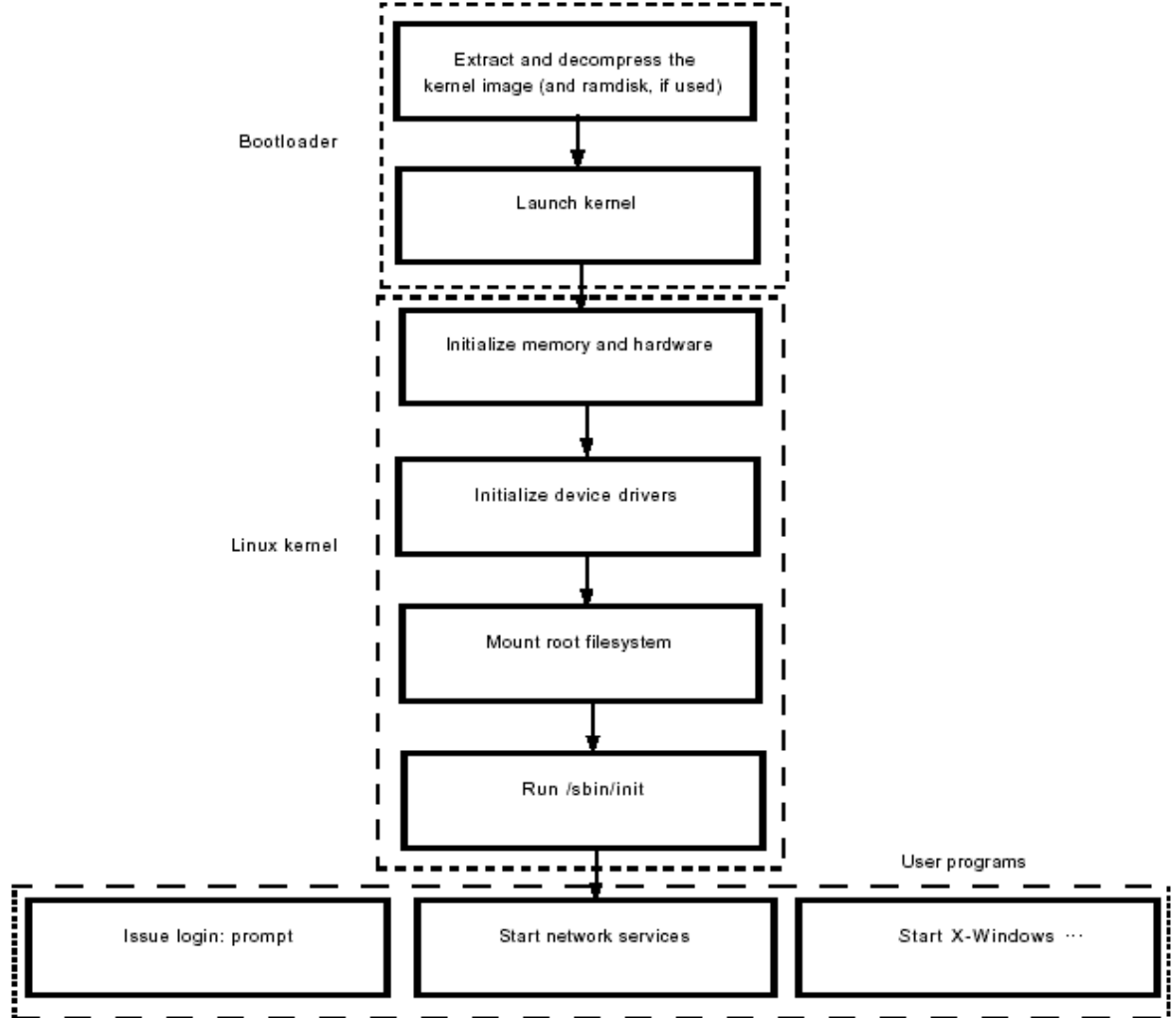
3.1 Çekirdek (Kernel)

Bir Linux sisteminde birçok proses değişik işler için atanmışlardır. Her proses sistem kaynaklarını kullanmak ister. Çekirdek tüm bu istekleri yerine getirmek amacı ile oluşturulmuş koddur [10],3. Çekirdeğin görevleri aşağıdaki bölümler doğrultusunda ayrımlanmıştır :

- **Proses yönetimi** : Proseslerin yaratılması, yok edilmesi, giriş çıkış işlemleri, birbirleri ile olan iletişimleri (IPC), çekirdek tarafından ele alınır. Ayrıca zamanlayıcı (scheduler), proses yönetiminin bir parçasıdır ve sistemde kritik bir öneme sahiptir [10],3.
- **Hafıza yönetimi** : En büyük kaynak hafızadır ve sistem performansının en önemli etmenidir.
- **Dosya Sistemleri** : Linux her şeyi dosya olarak görür. Bu nedenle çekirdek dosya sistemi üzerine kurulmuştur. Çekirdek birçok dosya sistemini tanımakta ve geliştiriciye cihazlardan yalıtılmış bir ortam sunmaktadır [10],4.
- **Cihaz Kontrol** : Hemen hemen her operasyon, fiziksel bir cihaz üzerinde gerçekleşir. Cihazlar, cihaz sürücüler (device driver) tarafından yönetilirler. Kullanılacak olan her cihaz için yazılmış bir sürücü, çekirdekte bulunmalıdır [10],4.
- **Ağ yönetimi** : Ağ yönetimi proseslerden bağımsız bir işlemdir. Bu nedenle çekirdek tarafından yönetilir. Gelen paketler asenkronudur. Bu paketler alınmalı, sıraya konulmalı ve ayrıştırıldıktan sonra, proseslere sunulmalıdır [10],5.

3.2 Çekirdeğin başlaması

Elektrik ile kavuştuktan sonra, bir gömülü sisteme kurulu bulunan Linux işletim sistemi, aşağıdaki şekilde görüldüğü gibi başlar. Burada başlamanın birbirinden bağımsız fazlardan oluştuğu görülmektedir. Çekirdeğin başlaması donanım birimlerinin başlatılması ile başlar ve /bin/init dizin sistemine dallanılması ve sonrasında kullanıcı programlarının başlatılması ile son bulur [20].



Şekil 3.2 Linux çekirdeğinin başlama prosedürü

Akış diyagramından da görüldüğü gibi “Bootloader” çekirdek imajını RAM e açar ve ardından çekirdek koduna dallanır. Bootloader aslında Linux’un bir kodu değildir. Bootloader, aslen kendi başına bir programdır ve donanıma bağlı bir kod parçasıdır. Öncelikle çekirdek kodunu RAM’e alır, kodu açar (decompress) ve bu kod içerisinde bulunan `_stext` yordamına dallanır.

Bootloader, çekirdek imajını hedef donanıma göre belirli bir lokasyona açar. Stext_yordamı ise çekirdek taban adresinden 0x1000 byte sonrasına yerleştirilmiştir. Örnek bir donanımda eğer RAM 0x8000 0000 adresinde bulunuyor ise çekirdek bu adrese açılır ve stext_yordamı 0x8000 1000 adresinde yer alır. Böylece bootloader bu adrese dallanmak durumundadır [20].

Stext yordamı donanıma ilişkin işlemleri yapar, yığın göstericisini kurar vb., burada temel, kodu minimumda tutmaktır. Stext işini bitirdikten sonra start_kernel() yordamına dallanır. Kesme rutinleri bu yordamda düzenlenmekte ve kesmeler başlığı altında anlatılmaktadır. Bu yordam /init/main.c dosyasında bulunur [20].

Start_kernel yordamının içerisine bakıldığında aşağıdaki çağrılar yapıldığı görülmektedir [20] :

- Setup_arch() : Donanıma ilişkin yapılandırmayı yapar. RAM in nerede ve hangi boyutta olduğunu tespit eder. Sayfalama işlemlerini de başlatır.
- Trap_init() : Bazı işlemcilerin kesme sistemini düzenler. Aslen işlemcinin kesme vektörlerini gerçek vektörleri göstermesini sağlar.
- Init_IRQ() : Linux çekirdeğinin kesmeleri yönetmesi ile alakalı düzenlemeleri yapar.
- Time_init() : Donanımın zamanlayıcı tiklemesini ayarlar. Zamanlayıcı kesmesini düzenler. Düzenleme sonrasında zamanlayıcı, her tiklemeye do_timer_interrupt() rutinini çağırır.
- Console_init() : Çekirdek başlatılmasından önce çağrılan bu yordam ile seri cihazlardan, bir takım veri iletişiminin kurulması amaçlanır. Açılış esnasında çekirdeğin vermiş olduğu mesajları seri cihaza aktarır. Bu sistem gerek Motorola gerek ise Intrinsyc kartta mevcuttur. Yalnız bu noktada ağ ile alakalı işlemler gerçekleştirilemezler. Zira ağ yöneticisi henüz çekirdek tarafından başlatılmamıştır.
- Init_modules() : Çekirdek modül sistemindeki “nsyms” parametresini belirler.

- `Calibrate_delay()` : Değişik mikroişlemcilerde bulunan gecikme rutinlerinin kabaca eşdeğer tutulmasını sağlamak için `BogoMips` ismi verilen hesabı yapar. `BogoMips` değeri, Linux zamanlayıcısı bahsinde de anlatacağımız “jiffies” değerine bağlı olacaktır. Eğer sistem zaman tıklaması çalışmıyor ise Linux açılışı bu noktada kilitlenecektir.
- `Mem_init()` : Çekirdek bellek yönetim sistemini düzenler. Ayrıca kullanıcı için hazır olan bellek miktarı ve çekirdeğin kullandığı bellek miktarını tespit eder.
- `Fork_init()` : `fork()` çağrılarının miktarlarını ve değişkenlerinin boyutlarını belirler.
- `Buffer_init()` : Çekirdekteki cache tamponunu düzenler. Bu alan birleşik dosya erişimlerinde ve sabit disklerden okuma ve yazmaları hızlandırır.
- `Page_cache_init()` : Çekirdekteki sayfa yönetim sisteminin cache’ini düzenler.
- `Signals_init()` : Çekirdekte bulunan sinyallere ait kuyruğu düzenler.
- `Proc_root_init()` : Linux’un /proc dizinini düzenler.
- `Ipc_init()` : IPC için var olan kaynakları düzenler.
- `Check_bugs()` : Donanıma özel kod bölümünde bulunur. Bu çağrı ile prosesor üzerinde bilinen hatalardan kaçınmak amaçlanmıştır. Mümkün olduğu müddette, hataların çevresinden dolaşmayı sağlayan kodlar bulunur.
- `Rest_init()` : Init fonksiyonlarının kullandığı belleği boşaltır ve `init()` prosesini bir çekirdek görevi olarak başlatır.
- `Init()` : Bu yordam `do_basic_setup()` çağrısı ile PCI ve ağ yönetimini başlatır ve böylece boot işlemini tamamlamış olur. Bu noktadan sonraki alınan bellek için bir şey yapılamaz, zamanlama başlatılmıştır, standart giriş çıkış ve hata sistemi başlatılmıştır, bu aşamadan sonra `prepare_namespace()` çağrılarak root dosya sistemi bağlanılır. Dosya sistemi bağlandıktan sonra “/sbin/init” programını başlatmak için “execve” çağrısı yapılır. Eğer

işletilecek bir program bulunamaz ise sırası ile “/bin/init” ve sonrasında “/bin/sh” çalıştırılmaya çalışılır eğer çalışacak bir “shell” oluşturulamaz ise “kernel panic” durumu oluşur ve çekirdek durur. Tüm bu durumlar aşağıda bulunan “init()” yordamı içeriğinden takip edilebilir :

```
static int init(void * unused)
{
    lock_kernel();
    do_basic_setup();

    prepare_namespace();

    free_initmem();
    unlock_kernel();

    if (open("/dev/console", O_RDWR, 0) < 0)
        printk("Warning: unable to open an initial console.\n");

    (void) dup(0);
    (void) dup(0);

    if (execute_command) execve(execute_command, argv_init, envp_init);
    execve("/sbin/init", argv_init, envp_init);
    execve("/bin/init", argv_init, envp_init);
    execve("/bin/sh", argv_init, envp_init);
    panic("No init found. Try passing init= option to kernel.");
}
```

Şekil 3.3 init() yordamının içeriği

- Do_initcalls() : Çekirdek cihaz sürücülerini ve modüllerini başlatır. Bu modüllerin başlatılabilmesi için __initcall mekanizması ile kayıt edilmesi gereklidir. Bu konuda ayrıntılı bilgi [20] kaynağında mevcuttur.
- Mount_root() : root dosya sistemini bağlamaya çalışır. Linux root dosya sistemini sabit diskten, floppy diskten veya nfs kullanarak ağdaki herhangi bir dizinden bağlayabilir. Ayrıca ramdisk de bu amaç için kullanılabilir. Linux çekirdeği panik yaratmadan önce birkaç kaynağı deneyecektir.
- “/sbin/init” programı : Bu program “init()” yordamından çağrılır. Bu programa bu noktadan sonra “init” ismi verilecektir. Bu program tüm proseslerin atasıdır. Görevi “/etc/inittab”da bulunan kullanıcı script’lerini çalıştırmaktır. “init” çalışmadan önce teknik olarak tüm “boot” işleminin

tamamlandığı varsayılır. Bu yüzden “init” programı bir kullanıcı prosesi olarak kabul edilir. “inittab” içeriği değiştirilerek veya “/etc/rc.d” dizinindeki alt dizinler değiştirilerek çekirdek tekrar derlenmeksizin, sistem davranışı değiştirilebilir.

3.2.1 Kuyruklar (Queues)

Linux çekirdeğindeki birçok kuyruk yapısı oluşturulmuştur. Bu kuyruklar zamanlama amaçları ile kullanılmaktadır [7]. Bazı kuyruklara örnekler [7] :

- Prosesler çeşitli nedenlerden dolayı bloke edilebilirler. Bloke edilmiş prosesler kuyruklarda tutulur. Çalışma zamanı geldiğinde proses tekrar çalışmak üzere kuyruktan çekilir.
- Olaylar (events), sırası geldiğinde işletilmeleri için FIFO kuyruklarında bekletilirler.
- Kesmeler de olaylara benzer şekilde kuyruklarda tutulurlar ama farklı olarak FIFO kuyruklarında değil öncelikli kuyruklarda tutulurlar.

3.2.2 Zamanlayıcı (Timer)

Linux çekirdeği zaman aralıklarını tutmak için bir zamanlayıcı kesmesi kullanır. Zamanlayıcı frekansı, tüm Linux sistemlerinde Hz cinsinden ifade edilir. Frekans değeri, <linux/param.h> dosyasında HZ sabitince belirlenir. Değeri, 100 olarak belirlenmiştir [10],131. Böylece sistem her 10 MS de bir kesme üretir.

Linux'ta sistem açılır açılmaz bir “unsigned long” değişken olan “jiffies” değişkeni, her kesmede bir artmaya başlar. Böylece sistem açıldıktan sonraki zaman bu değişken üzerinde ölçülmüş olur. “jiffies” değişkeni bu şartlar altında yılda bir sıfırlanır [10],131.

Eğer istenirse HZ değeri değiştirilebilir [7,10]. Bu durumda kullanıcı seviyesinde bir şey değişmez; fakat sistemin davranışında bir takım değişimler olabilir. Bu değişimler sistemin cevap verme hızını değiştirebilir. Ayrıca “jiffies” değerinin taşma süresi de otomatik olarak değişir [10], 132.

Bunlara ek olarak, eğer HZ değeri değiştirilirse, bu durumda tüm modüllerin tekrar derlenmeleri gerekir. Zira bu sabit, tüm modüllerde de kullanılmaktadır [10], 132.

3.2.3 Seviyeler

İşletim sistemleri teorisinde, değişik uygulamalar değişik alanlarda koşmalıdır. İşletim sisteminin görevi, donanımın sağladığı özellikleri, bağımsız operasyonlar için programlara sunmaktır. Bu işlemi gerçekleştirmeye çalışırken yetkisiz birtakım prosesler, değişik kaynaklara erişmek isteyebilir. Bunun önlenmesi gereklidir. Bu amaç doğrultusunda CPU'lar da bu tip işlemler için bir takım olanaklar sağlamalıdır [10],16.

Günümüz birçok CPU'su güvenlik önlemleri nedeni ile çeşitli program seviyeleri sunmaktadır. Her seviyenin kendine özgü bir rolü vardır. Bu rol içerisinde bazı işlemlere izin verilmeyebilir. Bir program kodu sadece belirli kapılardan geçerek bir seviyeden diğer bir seviye geçebilir [10],16.

Her ne kadar günümüz işlemcileri bir çok seviyeye sahip olsalar da (Intel x86 için 4 adet) Linux işletim sistemi sadece 2 seviyeyi desteklemektedir. Bu seviyeler : Kullanıcı (User) ve Supervizör (Supervisor) seviyeleridir. Çekirdek kodları supervizör seviyesinde koşup her türlü işlemi gerçekleştirebilirler iken kullanıcı programları kullanıcı seviyesinde koşarlar ve donanıma doğrudan erişemezler [10],16. (DOS işletim sistemindeki “inport”, “outport” gibi komutlar Linux işletim sisteminde bulunamazlar. Gerçekten de Microsoft da ileri versiyon işletim sistemlerinde, bu teknikleri tamamı ile kaldırmak durumunda kalmıştır.

Linux'ta kullanıcı seviyesinden süpervizör seviyesine geçiş ancak sistem çağrıları ve donanım kesmeleri ile mümkün olabilir. Sistem çağrıları proses içerisinde, sadece o proses ait olur iken donanım kesmeleri tüm sistemi ilgilendirdiğinden, sadece tek bir prosesin tekelinde değildir [10],17.

3.2.4 Kesme (Interrupt)

Kesmeler gömülü bir sistemin vazgeçilmez unsurlarıdır. Ayrıca kesmeleri bilmeden Linux işletim sisteminin bir platforma uyarlanması mümkün olmayacaktır. Linux işletim sisteminde kesmeler, “init/main.c” dosyasında bulunan “Start_kernel()”

yordamından başlatılır. Bu bölüm çekirdek kodunun donanıma bağımlı olan bölümündedir [19].

Donanım cihazlarından gelen bir kesme, çalışan prosesi sistemden çıkartır ve kendine ait bir kesme ele alıcısının çalışmasını sağlar. Burada düşük öncelikli bile olsa bir kesmenin, yüksek öncelikli bir prosesi sistemden çıkartması mümkündür. Bu durum bazı gerçek zaman kriterlerine uygun olmaz [7]. Genel olarak kesmeler kesme rutinleri içerisinde kapatılır. Bu amaç doğrultusunda cli() ve sti() fonksiyonları vardır. Böyle bir durumda, uzun kesme rutinleri, diğer kesmelerin alınamamasına neden olabilir. Bu durumu çözmek için, kesme rutini çalıştırmadan önce, kesme isteği kuyruğa alınır, daha sonra “thread” olarak çalıştırılır. Sonuç olarak, çalışan prosesin kesilmesi önlenir. Kuyrukta bulunan kesme isteği aşağıdaki durumlarda çalıştırılır [7] :

- Proses zamanlama “context switching” işlemi esnasında bir prosese görev verilmeden önce işletilir.
- Sistem çağrılarında dönüşte işletilir.
- Diğer bir kesmeden dönüşte işletilir.

3.3 Modüller

Linux çekirdeği çalışır durumda iken dahi genişletilebilir. Bazı kodlar çekirdeğe eklenebilir veya çekirdekten çıkartılabilir. Çekirdeğe eklenen her bir kod parçasına modül denir. Modüller obje kod (çalıştırılabilir dosyalar olarak değil) olarak tutulurlar. Çekirdeğe herhangi bir anda eklenmeleri ve çıkarılmaları için sırası ile “insmod” ve “rmmod” komutları kullanılır [10],5.

Modüller yığın işlemcide değişik biçimlerde kullanılabilir. Tuş takımı sürücüsü (Keyboard driver), görüntü sürücüsü (GUI driver) platforma ait bulunan USB gibi cihazların sürücüleri modül olarak tasarlanabilir. Böylece daha esnek, hızlı ve yönetilebilir bir sistem oluşturulabilir.

Modüller aynı zamanda sistem tasarımında yol göstermişlerdir. Bir modülde giriş, sonuç ve ana program kodları mevcuttur [10],15. Aynı durum ileride tasarım

esnasında algoritmalar için de söz konusu olacaktır. Burada modül yazarı çekirdekten tam anlamı ile yalıtılmıştır.

Modüller çekirdek alanında koşarlar [10],16.

3.4 Prosesler

Prosesler Linux işletim sisteminin bir işi böldüğü parçalarıdır [21], 283. Her parça ayrı bir proste bulunur ve işletim sistemi, işlemci üzerinde oluşturduğu zaman bölümlerini (time slices), proseslere paylaştırır. Her proses işletim sistemi tarafından tekil bir belirleyici (ID) verilir. Bu belirleyiciler aynı program başlatılsa bile, bir sonraki çalışmasında farklı olabilir. Bunun nedeni ise verilen ID'lerin sıra ile artan, bir tekil sayı olarak verilmesidir [21], 284.

Bazı prosesler işletim sistemine mahsustur ve bu proseslerin ID'leri sabittir. Fakat bu tip proses sayısı 5 i geçmez. En önemli sistem prosesi, 0 ID'ye sahip olan sistem zamanlayıcısı (system scheduler) dır. Bu prosesin görevi, zaman paylaşımını prosesler arasında yapmaktır. Bu proses aslında işletim sisteminde bulunan herhangi bir program değil, çekirdekte bulunan bir program parçasıdır [21], 285.

1 Numaralı proses “init” prosesidir. Bu prosesin görevi, işletim sistemini başlatmaktır. “/etc/rc” dosyalarını çalıştırır ve sistemi çok kullanıcı bir duruma geçirir. Eğer “init” prosesi “exit()” işlemini yaparsa yani bu proses yok edilirse bu durumda sistem kapatılmış olur [21], 285.

2 Numaralı proses, sanal bellek yönetimi olan sistemlerde sayfa yönetimini yapar ve “pagedaemon” ismini alır. Sayfa yönetimi, bellekte bulunan ve kullanılmayan sayfaların diske atılması sağlar. Böylece bellekte boşalan yerleri diğer proseslerin kullanması mümkün olur [21], 285.

3.4.1 Proses durumları

İşletim sisteminde prosesler değişik durumlarda bulunabilirler. Bu durumlar prosesin çalışıyor olması, uyuyor olması, zombi olması vb. dir.

3.4.2 Proses zamanlamaları

Prosesler çekirdekte bulunan bir rutin ile (“schedule”) zamanlanırlar. Unix zamanlayıcısı, CPU zaman parçacıklarını sistemdeki proseslere paylaştırmakla görevlidir. Bu amaçla her bir proses için bir öncelik hesaplanır. Bu hesaplama karışık bir şekilde ve sürekli olarak (çekirdekteki döngüde) yapılır. Hesaplama proses ile alakalı olan bir takım kriterler esas alınır [21], 290 :

- Proses ne kadar bellek kullanmaktadır ?
- Proses ne kadar giriş & çıkış yapmaktadır ?
- Proses en son aldığı anda ne oranda CPU kaynağı kullanmıştır ?

Bazı Linux işletim sistemlerinde değişik hesaplama yöntemleri kullanılabilir [7, 21], 290. Sonuç olarak proseslere birer öncelik verilir ve prosesler bu önceliğe göre sıralanır. Böylece yüksek öncelikli prosesler daha sık ve/veya uzun CPU zamanı alırlar [21], 290.

Proses öncelikleri çekirdek tarafından hesaplandığından, proseslerin doğrudan kendi önceliklerine müdahale etmesi mümkün olmamaktadır. Fakat “nice” ismi verilen bir değer sayesinde bir proses kendi için hesaplanan öncelik değerini bir miktar etkileyebilir. “Nice” değeri 0-40 arasında bir değerdir. Başlama değeri 20 dir. Değer 40 a yaklaştıkça prosesin önceliği düşer, tersine olarak 0’a yaklaştıkça proses önceliği artar. Nice değeri, “int nice(int incr);” yordamı ile değiştirilebilir.

3.5 Dosya sistemleri

Çekirdek kodu içerisinde birçok dosya sistemi mevcuttur. Çekirdek kodu derlenir iken bu kodlar da derlenmektedir. Dosya sistemleri çekirdek kodu içerisinde “fs” dizininde bulunmaktadır. Bazı önemli dosya sistemleri :

- Ext2 : “Second Extended Filesystem” olarak bilinir. Masaüstü bilgisayar sistemlerinde sıkça kullanılan dosya sistemidir [12]. Bu tezin geliştirilmesi ve uygulamaları sırasında, masaüstü PC sistemlerinde RedHat Linux Versiyon 7.2 kullanılmıştır. 7.2 şu an eski bir versiyondur ve kurulumunda varsayılan olarak “ext2” dosya sistemini kullanır.

- Ext3 : RedHat 7.3 ile birlikte gelen ve “ext2”nin ileri versiyonu olan dosya sistemi.
- Jffs : Flash File System. Özellikle gömülü sistemlerde bulunan Flash cihazlarını yöneten dosya sistemi. Burada önemli olan Flash’ın yazma politikasıdır. Flash yazma işlemlerinde 3 değişik sorun vardır :
 - Flash’ın ömrü yazıldıkça kısalır.
 - Yazma işlemi okuma işleminden uzun sürer.
 - Yazma işlemini yapabilmek için öncelikle silmek gerekir. Silme işlemi ise blok olarak yapılır.

Tüm bu sorunları çözmek için Jffs dosya sistemi geliştirilmiştir. Bu dosya sistemi sayesinde Flash’daki boş sektörler öncelikli olarak kullanılır.

- Ntfs : Windows NT işletim sistemine ait dosya sistemi.
- Fat : Dos işletim sisteminin dosya sistemi.
- Smbfs : Samba dosya sistemi. Samba Windows temelli işletim sistemlerinin Linux temelli işletim sistemlerinin dosya sistemini görebilmesi amacı ile oluşturulmuştur.
- Udf : “Compact Disc” teknolojisinin dosya sistemi.
- Nfs : “Network File System”. Ağ üzerinden, ağ üzerindeki diğer dosya sistemlerine erişmeyi ve bunları kendi dosya sistemi gibi görmeyi sağlayan bir dosya sistemidir. Fakat bu sistemin çalışması için Linux işletim sistemini kullanan sistemler gereklidir.

NFS dosya sisteminin geliştirme ortamlarında önemi büyüktür. Çalışma ortamlarında, hedef cihaza telnet veya PPP protokolü ile bağlanılır. Artık cihaz üzerinde her tür komut işletilebilir. Eğer başka bir sistemden veri alınması isteniyorsa bu durumda, ya FTP protokolü kullanılarak dosya transferi gerçekleştirilir ya da daha kolay olarak NFS’den faydalanılır. Cihazda bulunan bir dizine (genellikle /mnt altındaki bir dizine) geliştirmenin yapıldığı dosya sistemi

“mount” edilir. Bu amaç doğrultusunda “mount” komutu kullanılır. Gerek Motorola gerekse Intrinsyc kart bu şekilde çalışabilirler.

3.6 Ağ özellikleri

Ağ özellikleri günümüzde birçok gömülü sistemde desteklenmektedir. Ağ olarak genellikle Ethernet sistemi benimsenmiştir. Bu amaçla her üç kartta da CS8900 Ethernet sürücü entegresi kullanılmıştır. Bu entegre, Ethernet ağ arabirim entegresidir. Bu entegre sayesinde 10Mbit’lik bir Ethernet ağ sistemi kurulabilir.

Bugün kullanılan Internet altyapısı aslında Linux işletim sisteminin LAN yönetimi için kullandığı altyapıdır. Dolayısı ile Internet özellikleri aslında Linux içinde temel olarak içerilir. Internet denildiğinde ilk olarak akla gelen protokollerdir ve bu protokollerin en ünlüsü TCP’dir. TCP ile alakalı olarak ayrıntılı bilgi kaynaklarda mevcuttur [22], 544.

3.7 Bazı komutlar

I-Linux’taki komutlar, Linux’taki komutların bir alt kümesidir. Linux’taki temel komutlar, I-Linux’ta da korunmuşlardır [9]. Bu komutlar

- Dosya işleme için : ls, cp, rm, mv, less, cat, vim, chmod
- Dizin işleme için : pwd, cd, mkdir, rmdir, chmod, mount
- Prosesler için : ps
- Ağ işlemleri için : ssh, ping
- Zaman için : date
- Güç yönetimi için : shutdown, reboot.

şeklinde gösterilebilir. Bu listedeki komutlar ile çalışma platformu rahatlıkla yönetilebilir.

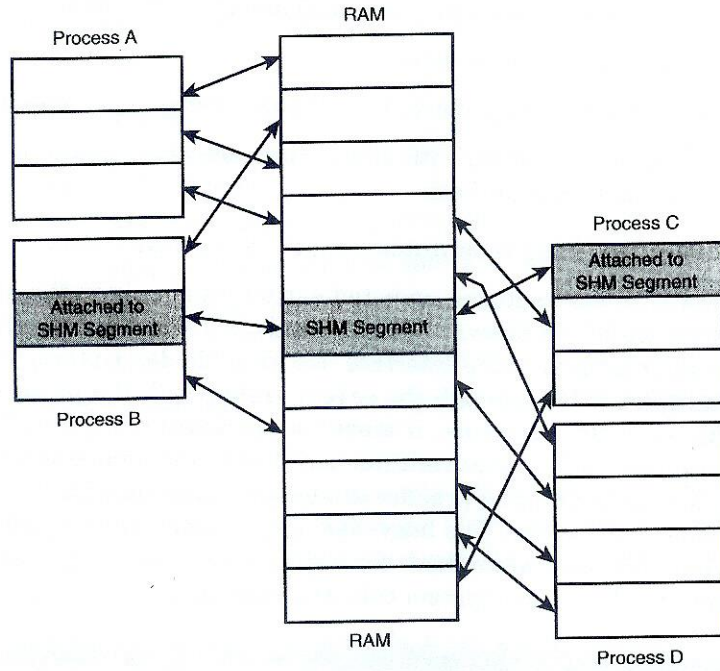
Komutların ne şekilde kullanıldıkları ve görevlerine ait doyurucu bilgi birçok kaynakta mevcuttur [11-13].

4. IPC – PROSESLER ARASI İLETİŞİM

IPC, işletim sistemlerinde bulunan prosesler arası iletişim metotlarının jenerik ismidir [8],319. Çok görevli işletim sistemlerindeki yaygın kullanımlarından dolayı POSIX IPC olarak standartlaştırılmıştır [8],340.

4.1 Paylaşılmış bellek (Shared Memory)

Paylaşılmış bellek çekirdekte bulunan ve prosesler arasında veri paylaşımını sağlamayı amaçlayan bir bellek alanıdır. Eğer yetkileri düzenlenirse diğer prosesler tarafından kendi veri alanları imiş gibi görünürler. Bir proses tarafından yaratılan paylaşılmış bellek bölgesi okunabilir, yazılabilir. Bir prosesin yazmış olduğu bilgi, paylaşılmış belleğe bağlanmış olan diğer prosesler tarafından da anında görülebilir [8],343.



Şekil 4.1 Paylaşılmış bellek kullanımı

Şekilde, 2 prosesin (B,C) ana bellekte yer alan bir paylaşılmış bellek bölgesine erişmesi gösterilmiştir. B,C prosesleri ana bellekte paylaştırılmış olan bir alana bağlanmışlardır. Dikkat edilirse burada bulunan 4 prosesin aynı zamanda fiziksel RAM üzerinde kendi veri alanları da bulunmaktadır ve bu alanlar diğerlerinden bağımsızdırlar [8],344.

Her bellek sayfası gibi bu sayfalar da MMU yönetimi sebebi ile diske atılabilir [8],344.

Paylaştırılmış bellek ile alakalı olan yordamlar şunlardır [8],350 :

- Shmget : Paylaşılmış bellek yaratır.
- Shmat : Paylaşılmış belleği bağlanmayı (attach) sağlar.
- Shmdt : Paylaşılmış bellekten ayrılmayı (detach) sağlar.

Paylaştırılmış bir bellek “attach” edildiği miktarda da “detach“ edilmelidir. “Detach” edilmemesi durumunda :

- “Attach” edilme sayıları sınırlı olduğu için özellikle bir döngüde yapılan “attach” işlemleri bir noktadan sonra çalışmaz hale gelir.
- “Attach” edilmiş bir paylaştırılmış bellek silinemez. Bu durumda bellekte gereksiz yere kullanılan paylaştırılmış bellekler yer kaplar. Özellikle büyük boyutlu değişkenler paylaştırılmış ise ana bellek bir noktada dolabilir.

4.1.1 Semaforlar

Semaforlar prosesler arasında veri değişiminde kullanılmaz. Birçok proses arasında paylaştırılan verilerin senkronizasyonunda kullanılan sayaçlardır. Paylaştırılmış bir veriye erişmeden önce proses :

- Veriye erişimi kontrol eden semaforu kontrol eder
- Eğer değer >0 ise proses veriyi kullanabilir. Kullanırsa semaforu 1 azaltır. Böylece bir kendisinin veriye eriştiğini bildirir.

- Eğer semafor değeri 0 ise bu veriye erişmek isteyen proses, semafor değeri 0 dan büyük oluncaya dek beklemeye alınır. Proses uyanınca ilk adıma geri döner.

Semafor yönetimi çekirdek tarafından yapılır [21], 385.

4.2 Sinyaller (Signals)

Sinyaller, IPC içerisindeki en basit formdur. Prosesler bu metodu birbirleri ile iletişim için kullanırken, çekirdek de prosesler ile iletişim için kullanır. Sinyal tanımı itibarı ile, proses için her an oluşabilecek, donanımdaki kesmelerin yazılımdaki tabiridir. Sinyaller her an gelebileceğinden dolayı asenkron olarak değerlendirilmelidir [8], 93, 94.

Her sinyalin bir ismi vardır. SIG ile başlar ve sonrasında mantıklı bir kısaltma gelir. SIGTERM (terminate sinyali), SIGSEGV (segmentation fault sinyali). Bu isimler <signal.h> dosyasında tanımlanmışlardır [8], 94. Bu konuda daha ayrıntılı bilgi aynı zamanda [8],96-98 numaralı kaynakta bulunmaktadır.

Bir proses sinyal oluştuğu zaman aşağıdaki üç işlemten birini yapabilir :

- Sinyali boş verir.
- Sinyali alır. Bu amaç doğrultusunda sinyal değerlendirici bir kod yazılmalıdır.
- Sinyalin tanımlı olan görevi var ise onu yapmasını sağlar [8], 94.

Bir sinyalin gönderilmesi bloke edilebilir veya gecikebilir.

Linux'ta bulunan zamanlayıcı da bir sinyal değerlendirici kod ile değerlendirilir. Zamanlayıcı için tanımlanmış olan sinyal "SIGALRM"dır [8],98.

Sinyaller "kill" fonksiyonu kullanılarak gönderilir. Sinyal elde ediciler ise aslında birer C fonksiyonudurlar ve sinyal seti ismi verilen yordamlar kullanılarak ele alınırlar. Sinyal setinde : "sigemptyset", "sigfillset", "sigaddset", "sigdelset", "sigismember" fonksiyonları bulunmaktadır.

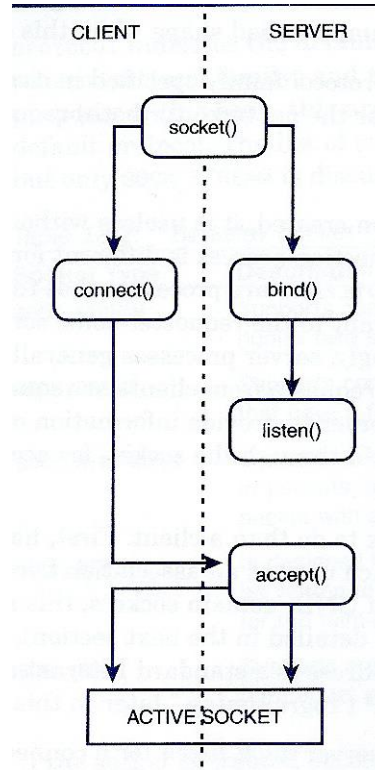
4.3 Soketler (Sockets)

IPC'den farklı olarak TCP soketleri, farklı makinelerdeki prosesler arasında iletişim sağlayabilirler [22], 543. Temel soket işlemleri “creating”, “opening”, “closing”, “reading” ve “writing” olarak belirlenmiştir. Linux'ta her şey dosya olarak görüldüğüne göre, sokette dosya olarak görülebilir ve dosyalar üzerindeki işlemler soketler üzerinde de çalıştırılabilir [8], 378.

Soket haberleşmesi ile alakalı olarak aşağıdaki işlemler yapılır :

- Socket() : Bu çağrı ile bir soket oluşturulur.
- Bind() : Sunucu, soketi oluşturduktan sonra bunu “bind” ile isimlendirir.
- Listen() : Bağlanma yapıldıktan sonra, sunucu, soketi dinlemeye başlar.
- Accept() : Eğer sunucunun dinlediği sokete bir istek gelir ise bu isteğin kabul edilmesini sağlar.
- Connect() : İstekçi tarafında çalışır ve yaratılan bir sokete bağlanılmayı sağlar.
- Send(), recv() : Sokete yazma ve okuma yapmak için kullanılır.

Yukarıda açıklanan soket API'si sayesinde, aşağıdaki şekilde görüldüğü gibi bağlantı kurulabilir [8], 382.



Şekil 4.2 Sunucu ve İstemci açılarından soket haberleşmesi

Sunucu (server) tarafından yapılan işlemler ile, istemci (client) tarafından yapılan işlemler arasındaki fark, yukarıdaki şekilde daha iyi anlaşılabilir. Sunucu istemciye göre daha çok işlem yapmaktadır. Sunucu tarafında özellikle kabul edilen her istek ayrı bir Linux prosesi tarafından değerlendirilmelidir. Böylece daha kolay bir programlama ortamına sahip olunur.

5. TASARIM

Tasarım bölümünde, öncelikle standart tasarım metodolojileri ve araçları açıklanacaktır. Ardından sistemin tasarımı, UML araçları kullanılarak yapılacak ve tasarım detayları anlatılacaktır.

5.1 Tasarım Modelleme Dili : UML

UML (Unified modelling language), nesne temelli analiz ve tasarım metodolojisi olarak duyurulduğu 1996 yılından beri oldukça rağbet görmüş ve kullanılmıştır. Sonuç olarak UML, nesneye dayalı modellemede de-facto standart olmuştur [14]. UML tasarımlarını temel alan birçok ticari ürün geliştirilmiştir. (Rational Software, Visio, IAR Visual State vb.).

UML hedef problem için değişik diyagramlar tanımlamaktadır [14] :

- Yapısal diyagramlar (structural diagrams)
 - Sınıf diyagramı (class diagrams)
 - Nesne diyagramı (object diagrams)
- Davranış diyagramları (behavioral diagrams)
 - Kullanım şart diyagramları (use case diagrams)
 - Sekans diyagramları (sequence diagrams)
 - Kolobrasyon diyagramları (collobration diagrams)
 - Durum geçiş diyagramları (state transition diagrams)
 - Aktivite diyagramları (activity diagrams)
- Uygulama diyagramları (implementation diagrams)
 - Aktivite diyagramları (activity diagrams)

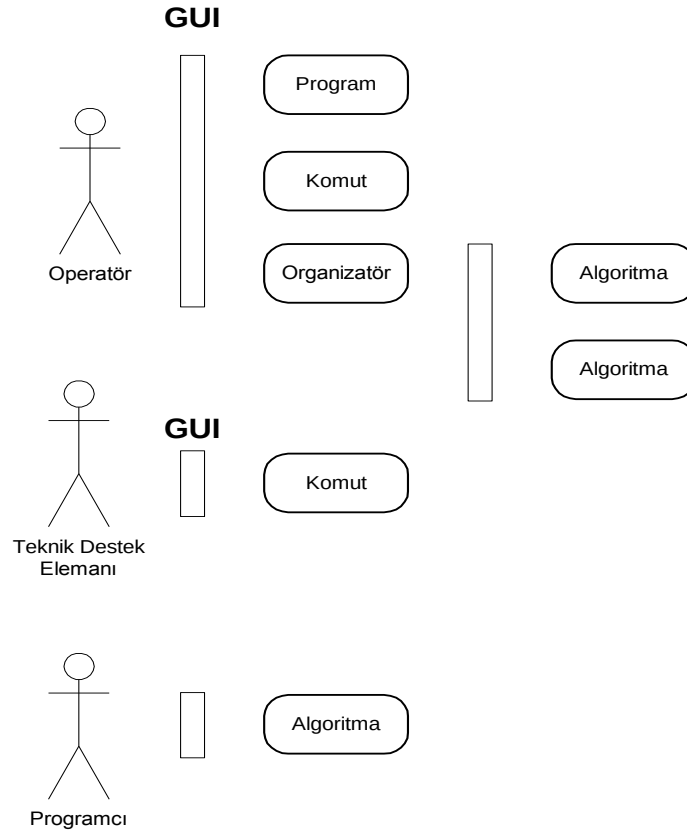
- Bileşen diyagramları (component diagrams)
- Sürüm diyagramları.(deployment diagrams)

Bu diyagramlar kullanılarak çok iyi bir yalıtım ile, karmaşık sistemler daha küçük ve birbirlerinden bağımsız kümeler şeklinde modellenebilirler. UML bu amaç doğrultusunda 2 adet araç sunmaktadır [14] :

1. Anlam (Semantics) : UML modelleme için bir sentaks ve anlamlandırma tanımlamıştır.
2. Notasyon (Notation) : UML model elemanları ve sunumları için grafiksel elemanlar tanımlamıştır.

5.2 Sistem aktörleri

Sunrise gibi bir sistem tek kişi tarafından kullanılamaz. Sistemde 3 adet kullanıcı tanımlanmıştır. Bu kullanıcılar aşağıdaki şekilde gösterilmiştir :



Şekil 5.1 Sistemde tanımlı kullanıcılar

Şekilde bulunan aktörler aşağıda açıklanmıştır :

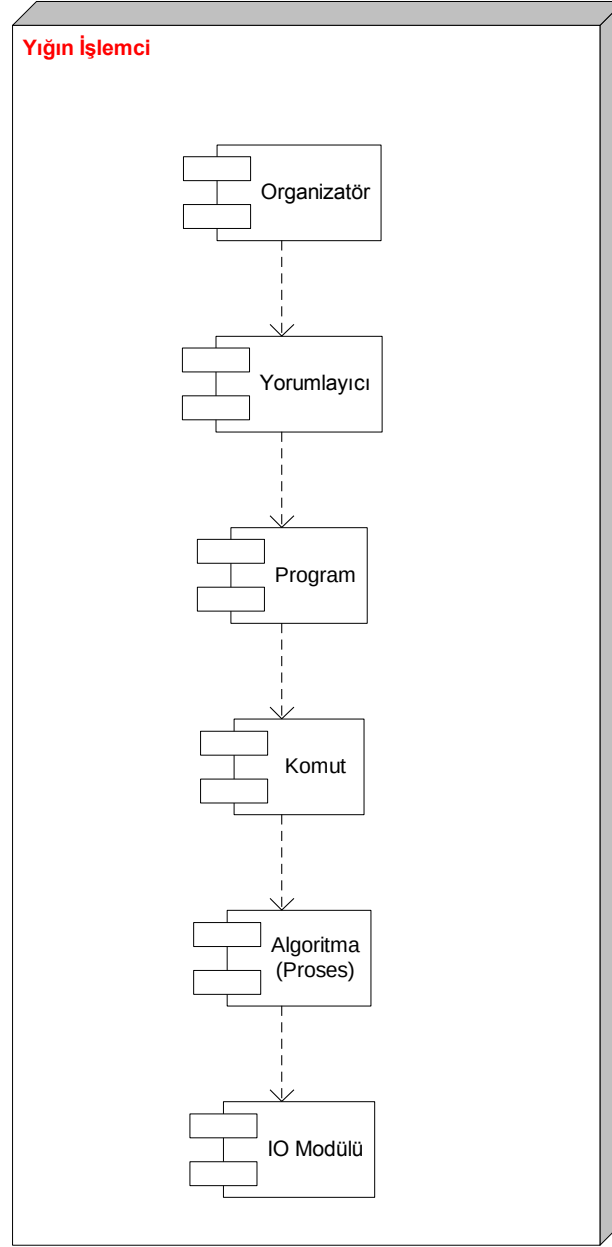
- **Programcı** : Sistemde bulunan algoritmaları yazar.
- **Teknik Destek Elemanı** : Algoritmaların komut şekline dönüştürülmesini sağlar. Bu işlem operatöre yüklenemeyecek kadar teknik ve cihaz spesifik bir işlemdir.
- **Operatör** : Cihazı endüstriyel ortamda kullanan kişidir. Bu kişi komutları kullanarak program yazar. Ayrıca organizatörü kullanarak programı kontrol eder.

5.3 Blok Diyagram

Şekilde yığın işlemciye ait UML bileşen diyagramı görülmektedir. Bu diyagramda sistemde bulunan modüller ve bu modülleri arasındaki bağlantılar gösterilmektedir.

Diyagramdan görüldüğü gibi sistem birbirinden bağımsız, 6 adet modülden oluşturulacaktır. Bu durum kodlamanın daha kolay ve yönetilebilir olmasını sağlar. Ayrıca modüller kendi içlerinde test edilip (black box testleri) sisteme entegre edilir ise sistemin daha sorunsuz bütünleşmesi ve toplamda daha az sorunlu olması mümkün olur.

Diyagramda en alt seviyeden yukarı seviyelere doğru çıkışta, donanıma yakın olan seviyeden kullanıcıya yakın olan seviyeye doğru gidilir.



Şekil 5.2 Sunrise, UML bileşen diyagramı

Her bir modülün ne işe yaradığı aşağıdaki şekilde özetlenebilir :

- IO Modülü : Algoritmaların ihtiyaç duyduğu giriş çıkış işlemlerini düzenler. IO API'si IO'lara erişimi düzenler. Ayrıca IO yönetimi algoritmaların yaptığı IO işlemlerini fiziksel olarak gerçekleştirmek ile sorumludur.
- Algoritma : Linux prosesleri olarak tasarlanacaktır. İşletim sistemindekine benzer şekilde her bir algoritmanın proste bir görevi olacaktır. Prosesin

ihtiyaç duyduğu ısıtma, soğutma, doldurma, boşaltma vb. işlemler algoritmalarda yapılır.

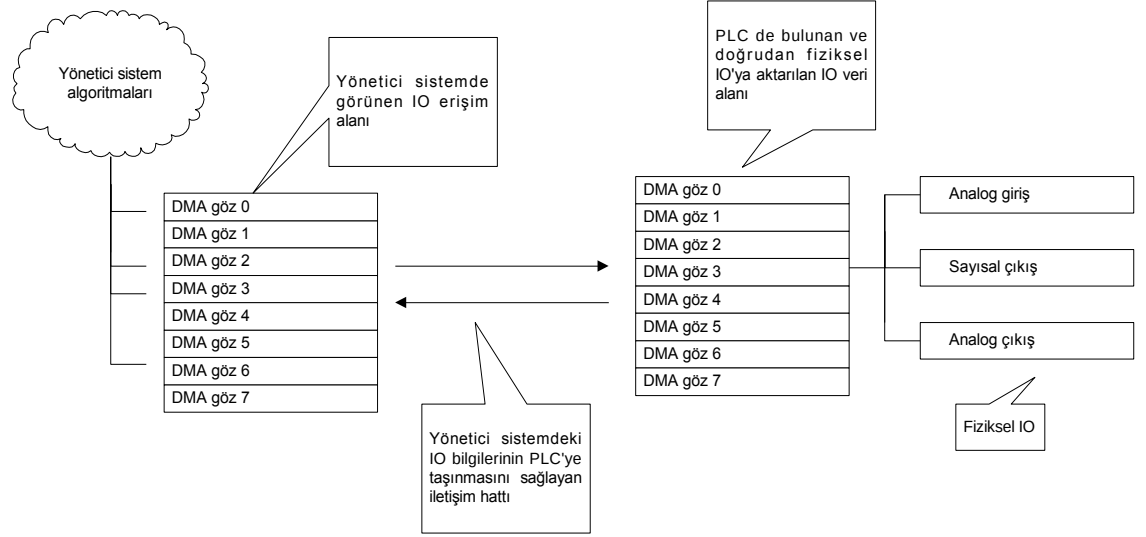
- Komut : Her bir algoritmanın parametreleri olacaktır. Isıtma için derece, doldurma için miktar vb. gibi. Bu parametreler ile algoritmaları birlikte düşündüğümüzde bir komut oluşturacaktır.
- Program : Komutların oluşturduğu bütün program olarak nitelendirilir. Program endüstriyel prosese ait olan tüm bilgiyi içermektedir.
- Yorumlayıcı : Programların yorumlanıp çalıştırılmaya hazır hale getirilmesi işlemidir. Bu sayede program çalışma ortamı oluşacaktır. Bu ortam paylaşılan bellek, çalışmaya hazır algoritmalar ve bunların koordinasyonunu sağlamakla görevli organizatörün oluşturulması ile oluşacaktır
- Organizatör : Organizatör çalışmaya hazır bir programı başından sonuna dek çalıştırır. Çalışma esnasında oluşan beklenmeyen durumları ele alır. Ayrıca paralel komutları da dikkate alır. Kullanıcıya yakın bir bileşendir. Kullanıcıdan gelen emirleri program üzerinde işletir.

5.4 IO Yapılandırması

Algoritmalar, Linux bölümünde proseslerin yaptığı gibi Sunrise’da da giriş çıkış yapmak zorunda kalacaklardır. PID algoritmasını örnek alıp bu algoritmanın sıcaklık kontrolü yaptığını varsayarsak, geri besleme döngüsünde sıcaklık bilgisi sensörden okunacaktır. Bu sıcaklık değerinin algoritmaya yani prosese gönderilmesi gerekir.

5.4.1 PLC IO yapılandırması

PLC’lerde IO yapılandırılması DMA’lar yardımı ile yapılır. DMA “array” yapısında bir veri alanıdır. Bu veri alanı IO sistemine paralel olarak aktarılır. Yani DMA alanındaki dizi bilgiler fiziksel IO’lara gönderilir. Bu durum aşağıdaki şekilde sunulabilir :



Şekil 5.3 Terminaldeki verilerin fiziksel giriş çıkış birimlerine aktarılması

5.4.2 Tasarım IO yapılandırması

IO sistemi tasarımı değişik şekillerde kurulabilir. Bu tasarımın nasıl yapıldığına bağlı olacaktır. IO sistemi algoritmaların ihtiyaç duyduğu giriş çıkış işlemlerini sağlamakla görevlidir. Bu amaçla IO sistemi algoritmaların IO işlemlerini fiziksel IO işlemlerine dönüştürecekler.

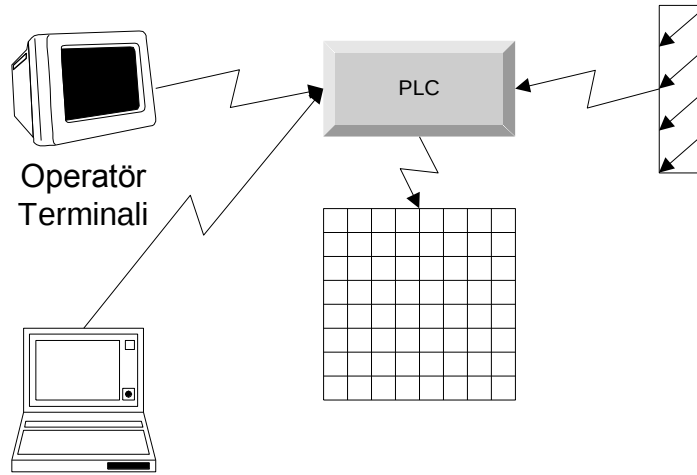
Genel olarak IO yapısı iki biçimde kurulabilir :

- PLC'dekine benzer bir dizi ("array") şeklinde. Eğer IO sistemi çok basit bir şekilde kurulmak isteniyorsa bu yapılabilir. Bu durumda C dilinin getirdikleri kullanılmamış olur. Bir dizi üzerinde bulunan her eleman aslında bir fiziksel IO değerine karşı düşmektedir. IO sistemi dizi değerlerini ve fiziksel IO'yu birbirlerine eşler. Böylece her dizi elemanı fiziksel bir IO'ya karşı düşmüş olur.
- C jenerik bir formda. Bu durum IO sisteminin biraz daha yetenekli olması durumunda kullanılır. Her IO kanalının bir nesne olduğunu düşünürsek bu tip bir tasarım gereklidir. Örnek olarak bir analog giriş kanalını düşünelim. Bu IO kanalının ismi, değeri, kalibrasyon bilgileri, DMA'daki yeri, birimi, minimum ve maksimum değerleri vb. Bir yapı "struct" içinde toplanabilir ve bu yapı doğrultusunda

incelenebilir. Böyle bir durum daha yetenekli IO sistemi elde edilebilir fakat sonuçta eğer PLC kullanılacak ise DMA alanında olduğu gibi düz bir “array”e çevrilmek zorundadır. PLC kullanılmayacak ve doğrudan fiziksel IO sistemine yazılacak ise DMA’da olduğu gibi “array” yapısına çevrilmesine gerek yoktur.

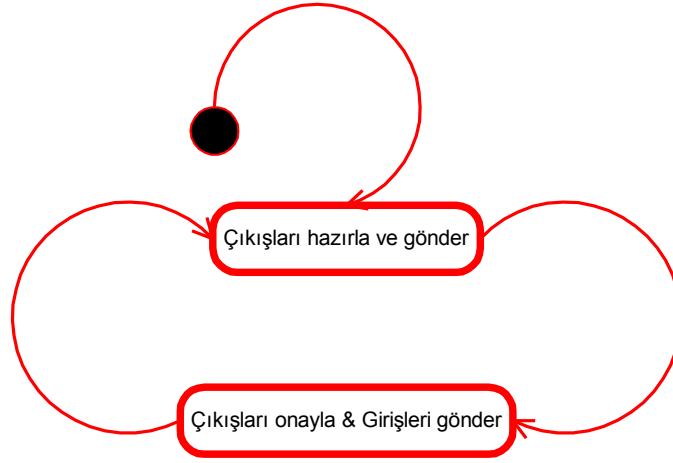
5.4.3 Önerilen PLC iletişimi Tasarım

PLC’lerin IO sisteminde kullanılması oldukça yaygındır. Bu bölümde PLC ve operatör terminali olarak 2 parçalı bir tasarım yapıldığı düşünülerek (Remote IO) bir takım iletişim önerileri sunulacaktır. PLC iletişimi genellikle RS232 ile yapılmaktadır. Fakat burada anlatılanlar arabirimden bağımsız olarak kavramsal bir biçimde yorumlanacaktır. İki parçalı tasarım aşağıdaki şekilde resimlendirilebilir :



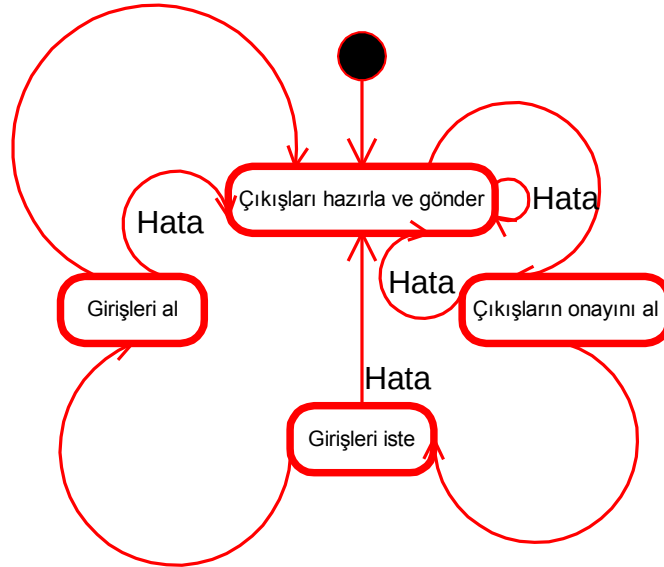
Şekil 5.4 İki parçalı tasarım

PLC ile iletişim sonlu durum diyagramı olarak düşünülmelidir. Eğer güvenilir bir iletişim tasarlanacak ise bu diyagram en az 2 durumlu olmalıdır. Aşağıda görüldüğü gibi ilk durumda IO’lar okunmalı ve yazılmalı diğer durumda ise okunan ve yazılan değerlerin OK bilgisi elde edilmelidir.



Şekil 5.5 2 Durumlu bir iletişim makinesi

Yukarıdaki sistemde IO okuma ve yazma paketi IO sayısına orantılı bir şekilde uzar. Bu durumda iletişim hataları artabilir. Böyle bir pozisyona girmemek için durum diyagramının durum sayılarının artırılması düşünülmelidir. Bu durumda durum sayısı 2'nin katları şeklinde büyümelidir. İkinci bir tasarımda 4 durumlu bir tasarım aşağıdaki gibi düşünülebilir :



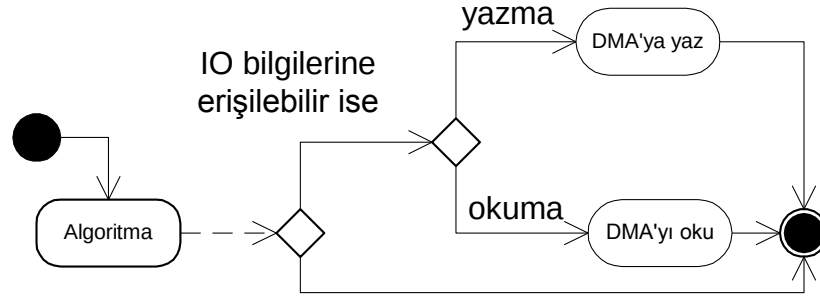
Şekil 5.6 4 durumlu bir iletişim makinesi – OMRON PLC iletişim protokolü

Benzer şekilde IO sayısının daha da artması dolayısı ile durum sayısı daha da artırılabilir. Bu sonucu değiştirmez ve sunulan sistem sorunsuzca uzatılabilir.

8 seviyeli IO durum diyagramında analog çıkışlar yazılır (1), yazılmanın onayı gelir (2), analog girişler okunur (3), okumanın onayı gönderilir (4), sayısal çıkışlar yazılır (5), yazılmanın onayı gelir (6), sayısal girişler okunur (7), okumanın onayı gönderilir (8).

5.4.4 IO Kullanılması

IO'ların algoritmalarca kullanılması bir API sayesinde yapılmalıdır. API ile IO sistemi arasındaki ilişkiler aşağıdaki şekilde yapılabilir :



Şekil 5.7 IO API'si use case diyagramı

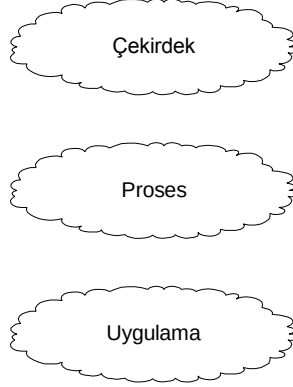
Görüldüğü gibi API de okuma, yazma ve kontrol fonksiyonları mevcuttur.

5.5 Algoritmalar (Prosesler)

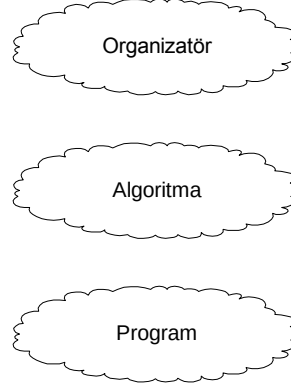
Algoritmalar yığın işlemcisinin en temel öğeleridir. Algoritmaya örnek olarak kontrol sistemlerinde en yaygın kullanılan PID algoritması gösterilebilir. PID algoritması endüstriyel sistemlerde genellikle her türlü kontrolde kullanılabilir. Isıtma, soğutma, doldurma vb. Temel olarak yaratılacak olan bir ayrık zamanlı PID algoritması, bir çok komutun temelini oluşturacaktır.

Linux işletim sistemi, proses temelli bir işletim sistemidir. Prosesler, Linux işletim sisteminin en temel öğeleridir. Bu nokta, algoritmaların, yığın işlemcinin temeli olması ile benzerlik göstermektedir. Bu durumda, aşağıdaki kavramsal düşünce geliştirilebilir.

LINUX

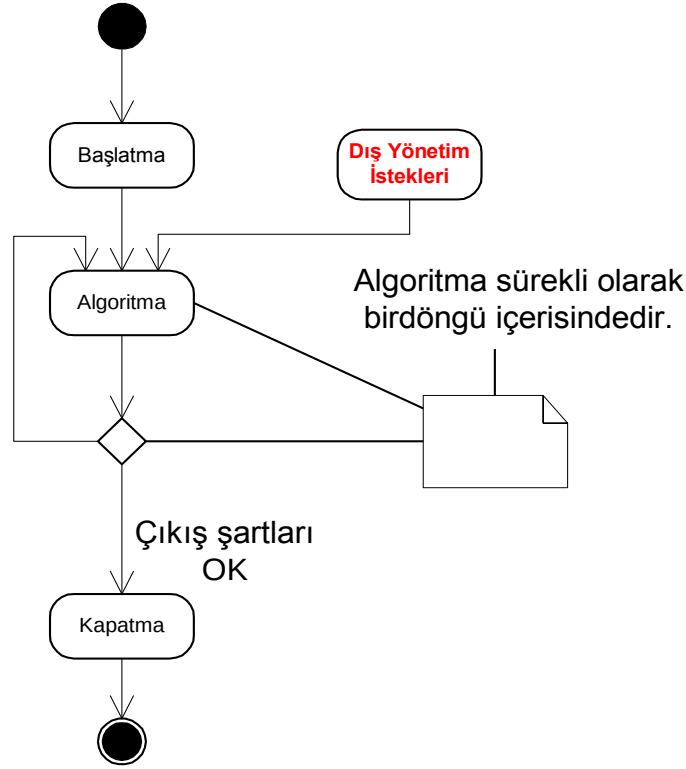


SUNRISE



Şekil 5.8 Linux ve Sunrise'ın benzerlikleri

Algoritmalar tüm platformdan bağımsız olarak C dili ile geliştirileceklerdir. Geliştirme, algoritmaların sistemin temel öğeleri olduğu ve modülerizasyonu da sağlayacağı için dikkatli bir şekilde Linux modüllerine benzer bir biçimde yapılması gereklidir. Bu amaç doğrultusunda aşağıdaki şekilde görüldüğü gibi bir akış diyagramı düşünülmelidir:

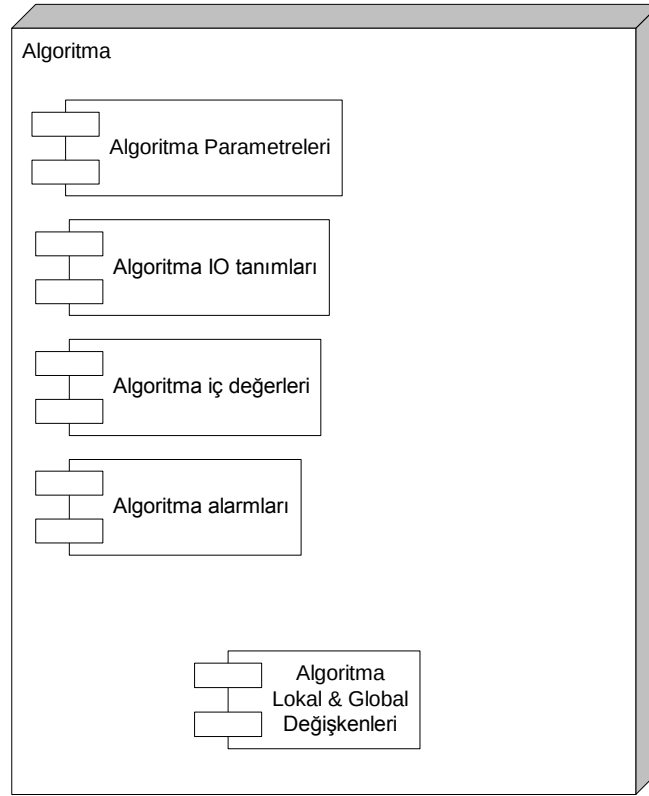


Şekil 5.9 Algoritma akış diyagramı

Akış diyagramındaki ilk adım olarak algoritmanın başlama değerlerinin düzenlenmesi (initialization) gereklidir. Düzenleme işlemi bir kereye mahsus olarak yapılır. Dolayısı ile algoritma bu bölümde dosya açabilir, değişkenlere yer alabilir, kullanacağı IO değerlerini düzenleyebilir.

Başlama bölümünün sonrasında, algoritma döngüsü başlar. Algoritma döngüsü sürekli bir döngüdür. Bu döngü esnasında, algoritma, amacına ulaşmaya çalışacaktır.

Algoritma çıkış şartları oluştuğunda başlatılan değerler eski durumlarına getirilir, alınmış bellek bölgeleri bırakılır, açılan dosyalar kapatılır ve algoritma bu şekilde sona erer.



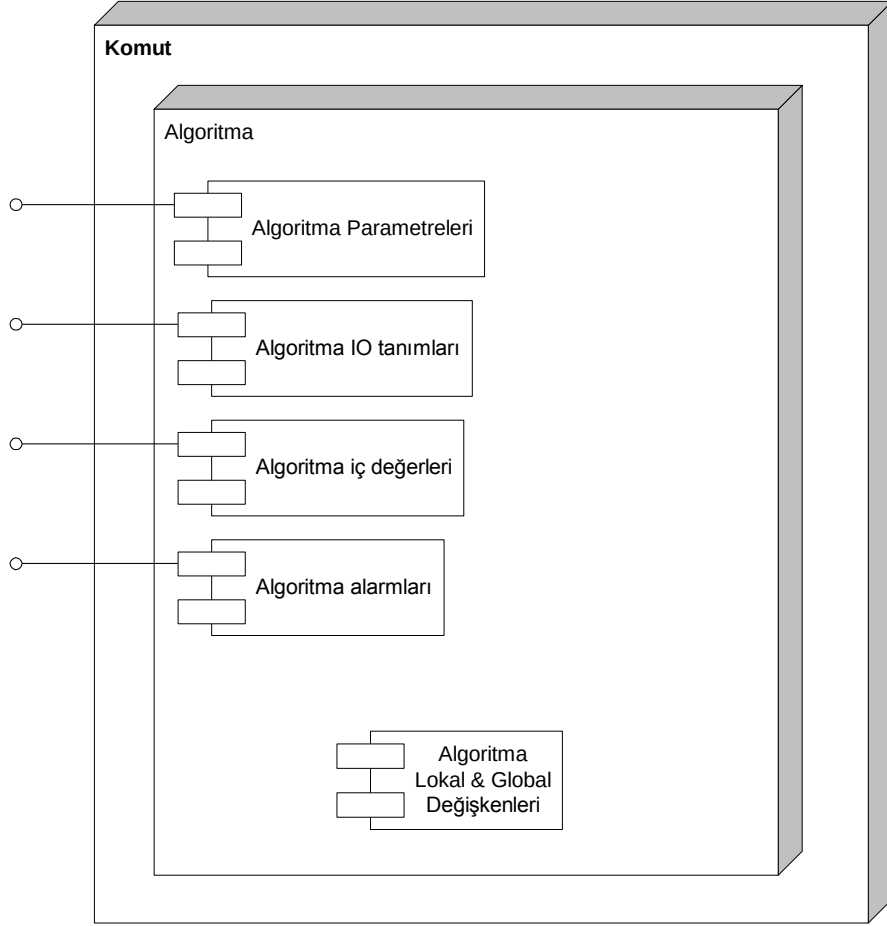
Şekil 5.10 Algoritma bileşenleri

Algoritmalar yukarıdaki şekilde görüldüğü gibi bir takım bileşenlerden oluşur. Bu bileşenler algoritmaya dışarıdan gelebilir veya algoritma tarafından dışarıya sunulabilir. Bu bileşenler başlıca aşağıdaki başlıklar altında incelenebilir :

- Parametreler : Algoritmaya dışarıdan sunulan hedef değerleri belirtirler. Sıcaklık kontrol algoritması için hedef sıcaklık, materyal doldurma algoritması için miktar vb. Ayrıca en önemli parametre zamandır. Birçok algoritma zamana ihtiyaç duyar. Algoritmanın vereceği çıkışlar zamana göre hesaplanarak verilir.
- IO'lar : Algoritmaya hangi IO'ları kullanması gerektiğini söyler. Sıcaklık kontrolünü düşünürsek 2 adet IO'ya ihtiyaç olabilir. Birisi ısıtmayı sağlayan rezistansı harekete geçirmek için diğeri ise gelinen sıcaklığı okumak için.
- İç değerler : Bazı durumlarda algoritmanın içinde neler olmakta olduğu bilinmek istenebilir. Bunun değişik nedenleri olabilir : Algoritma debug edilmek isteniyordur, algoritmanın nereye ulaşmak istediği bilinmek isteniyordur, kullanıcıya bilgi sunulması isteniyordur vb.
- Alarmlar : Algoritma olağan dışı durumlarda, çalışmasını sürdüremez veya kendine verilen görevi yerine getiremez. Böyle durumlarda algoritmanın bu durumları organizatöre bildirmesi gereklidir.

5.6 Komutlar

Yığın işlemcisi hiyerarşisi içerisinde algoritmaların üzerinde komutlar bulunmaktadır. Buradaki ilişki aşağıdaki şekilde gösterilebilir :

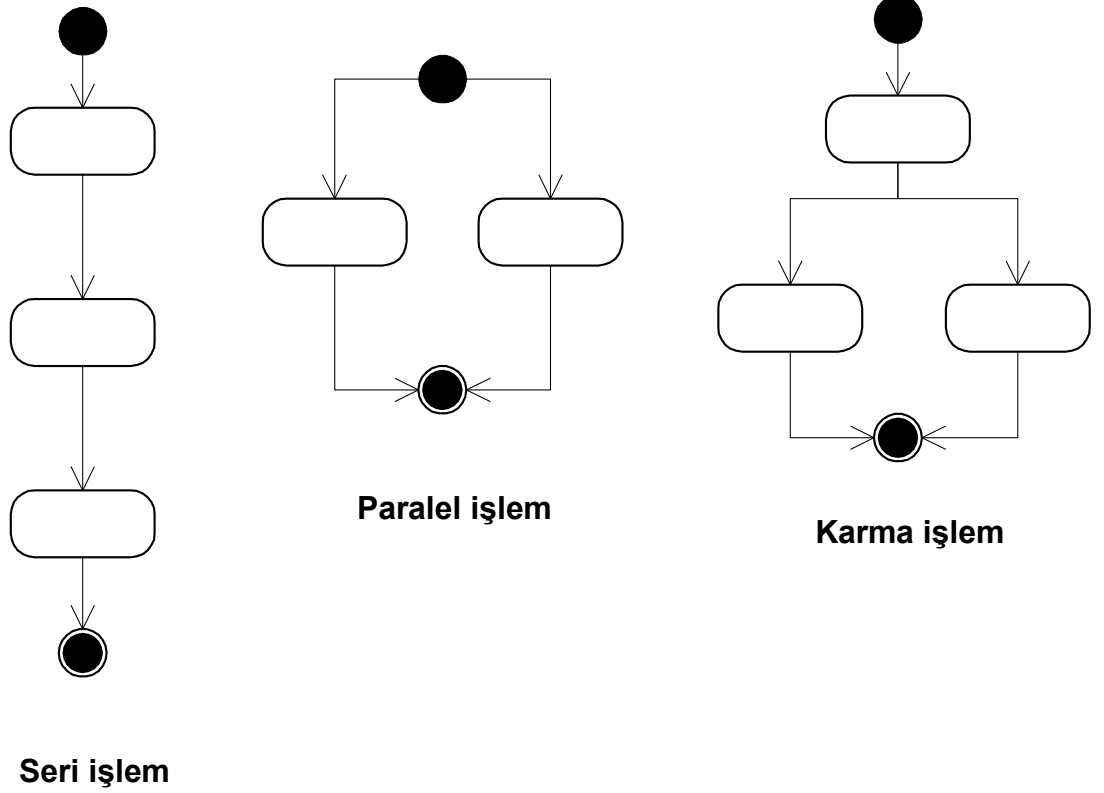


Şekil 5.11 Komutlar

Şekilde algoritmalarından farklı bir şey yoktur. Algoritmalar komutlara dönüştürülmüştür. Dönüştürme işleminde algoritmayı oluşturan içsel bileşenler dışarıya açılmıştır. Dolayısı ile algoritmaların kullanıcıya sunulan haline komut denilebilir.

5.7 Program (Yığın işlem)

Programlar bir veya birden çok komutun birleştirilmesi ile oluşturulur. Yığın işlemlerdeki komutlar seri, paralel ve karışık olmak üzere değişik pozisyonlarda bulunabilir. Bu durum yığın işlem standartlarını belirleyen dokümanlarda da gösterilmektedir :



Şekil 5.12 Yığın işlem standart adım pozisyonları

Bir işletim sistemi üzerinde programlar değişik şekilde saklanabilirler. Eğer veri tabanı kullanılıyorsa [15] bu durumda programlar veri tabanında saklanabilir. Böyle bir veritabanı [15], 150 KB gibi ufak bir kod alanına sığmıştır ve Linux üzerinde de çalışabilmektedir. Böyle veritabanlarında, masaüstü bilgisayarlardakinden farklı olan uyarlamalar yapılmıştır. Bu uyarlamalar, limitli bellekte çalışma ve sınırlı kayıt alanı bulunması üzerinde yapılmıştır. Bazı SQL ifadeleri eksiltilmiştir fakat bir veritabanının sağlaması gerektiği temel SQL ifadelerini sağlamaktadır [15] :

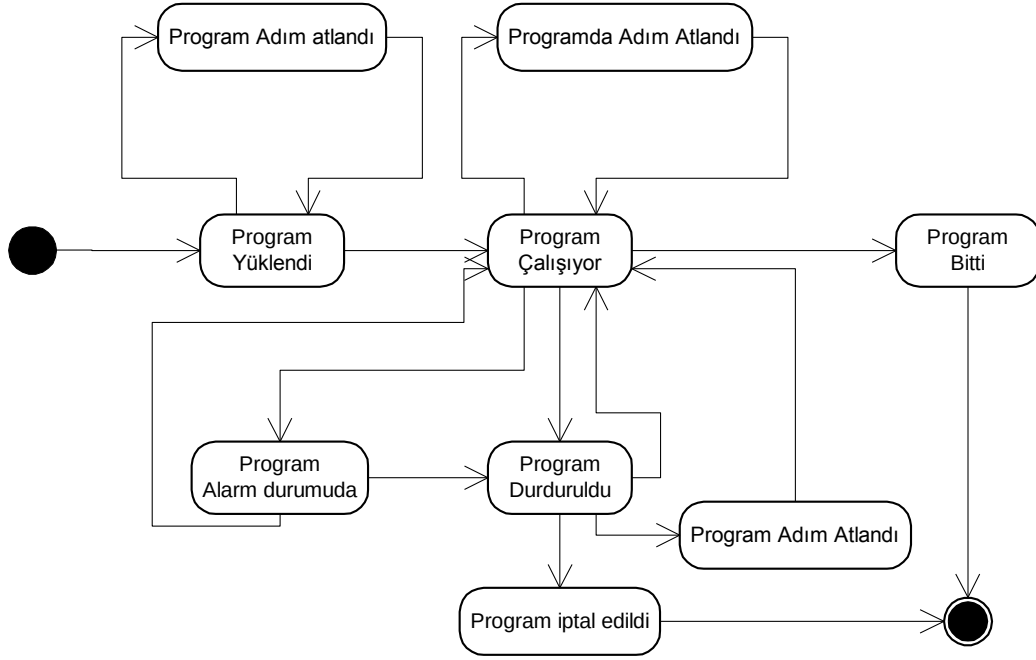
- Data tipleri : Integer, smallint, char, varchar, blob, time, date, decimal.
- SQL sorgulama : Select, from, where, group by, order by, limit...

Veri tabanı yok ise işletim sisteminin sunduğu dosya sisteminden faydalanılmak zorundadır. Fakat bu durumda da, veri tabanı tasarımında kullanılan bir takım kriterlerin [15] kullanılması uygun olacaktır :

- Dosya açma ve kapama pahalı bir işlemdir. Bundan sakınılmalıdır.

- Çok fazla dosya kullanılmamalıdır. Bunları yönetmek zordur.
- Gerekiyorsa cache kullanılmalıdır. Daha hızlı erişim sağlanabilir.
- “Memory mapped IO”, “stream based IO”dan daha performanslıdır. Her iki yöntem de Linux da desteklenmektedir. Uygulamada seçim yapılırken göz önünde bulundurulabilir.

Sistemde programlar değişik durumlarda bulunabilir :



Şekil 5.13 Program durumları

Program ilk durumda yüklendi pozisyonuna geçer. Bu durumda kullanıcı programı istediği adımdan başlatabilir. Program çalışıyor pozisyonunda devam eder. Eğer hiçbir sorun çıkmaz ise ideal durumda program biter.

Program çalışması esnasında kullanıcı veya başka bir nedenden ötürü durdurulabilir. Böyle bir durumda kullanıcı ya tekrar başlatacak ya da adım atlatacaktır. Program, alarm durumuna her an geçebilir. Hatta alarm durumunda iken bile sonlanabilir. Dolayısı ile program alarm verse dahi çalışmasına devam edebilir. Burada alarmin niteliği önemli olacaktır. Bazı alarmlar programın çalıştırılmasına engel olabilir.

5.8 Yorumlayıcı

Programlar Linux derleme işlemindeki aşamalarda olduğu gibi (preprocessing, compiling, assembling, linking) öncelikle yorumlanmalı ardından çalıştırılmalıdır.

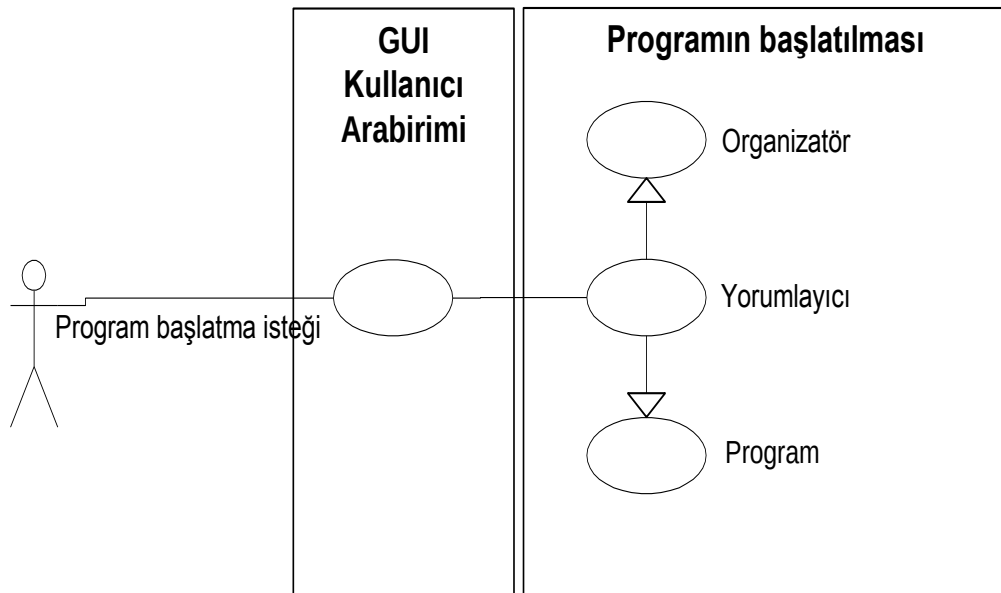
Yorumlama aşamasında program ile alakalı birçok şart kontrol edilir :

- Programdaki komutların içerdiği algoritma çalıştırılabilirleri sistemde bulunmakta mıdır ?
- Programdaki komutlar tanımlı mıdır ?
- Kullanıcı programı uygun bir şablon ile yazmış mıdır ?
- Kullanıcı algoritma parametrelerini sınırlar içinde göndermiş midir ?

Bunların dışında program ve sistem ile alakalı olarak bir takım çıkarsamalar da yapılabilir. Bu çıkarsamalar aşağıdaki bilgileri verebilir :

- Programda sistemdeki kaynaklar ne şekilde kullanılıyor ?
- Program optimize edilebilir mi ?

Yığın işlemcide program başlatmaya ilişkin diyagram aşağıdaki şekilde gösterilmiştir :

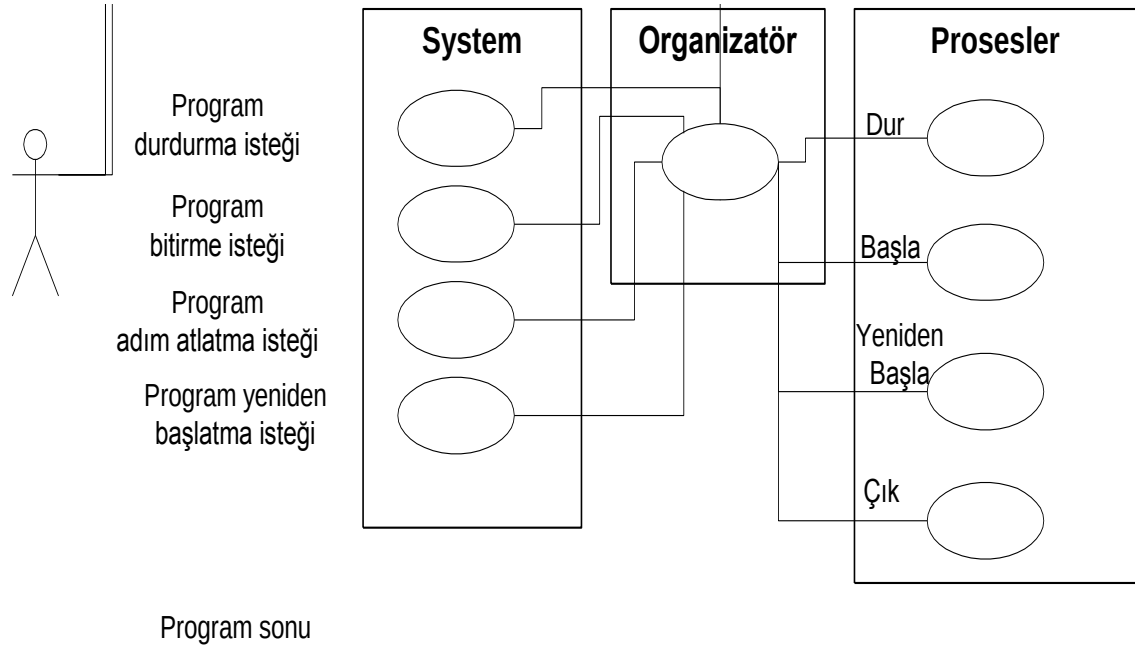


Şekil 5.14 Program başlatma prosedürü

Şekilde görüldüğü gibi kullanıcı GUI ile yüzleşmektedir. GUI tarafından alınan program başlatma isteği yorumlayıcıya bildirilir. Yorumlayıcı aldığı program bilgisi ile programa erişir. Programı yorumlar ve ardından organizatörü başlatır. Bu esnada sistem “Program Yüklü” durumuna geçmiş olur.

5.9 Organizatör (Çalıştırıcı)

Organizatör çalışma anında programı kontrol eden ve senkronizasyonu sağlayan bölüm olacaktır. Bu bölüm Linux’taki çekirdeğe karşı gelmektedir. Organizatör yorumlayıcı tarafından başlatılır.



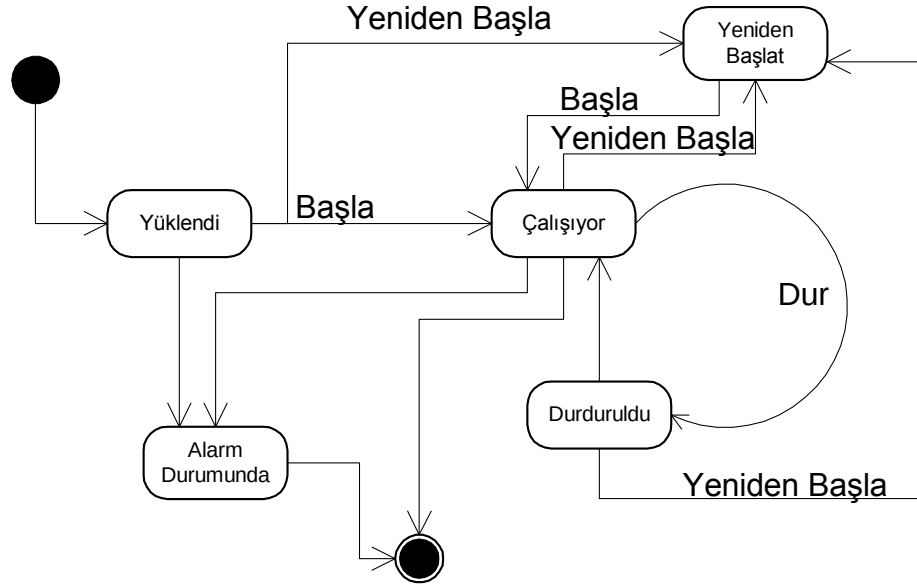
Şekil 5.15 Organizatör kullanım şart diyagramı

Organizatör, kullanıcı isteklerini GUI üzerinden alır. Ayrıca programın normal şartlarda bitmesi de organizatör tarafından bilinir. Tüm bu durumlara karşı düşen emirler algoritmalara iletilir. Organizatör ile algoritmalar arasındaki iletişim gerek organizatör ve tek bir algoritma arasında gerekse organizatör ve tüm çalışan algoritmalar arasında olabilir. Algoritmalara iletilen görevler 4 tane görünmektedir :

- Dur : Algoritmanın durması istenmektedir.

- Başla : Algoritmanın başlaması istenmektedir.
- Yeniden Başla : Algoritmanın içsel değişkenlerini tekrar başlatması istenmektedir.
- Çık : Algoritmanın sonlanması istenmektedir.

Tüm bu şartlar altında bir algoritmanın alabileceği durumlar ve bu durumlardaki geçişleri aşağıdaki şekilde gösterebiliriz :



Şekil 5.16 Algoritma durum diyagramı

5.9.1 Linux zamanlayıcısının kullanılması

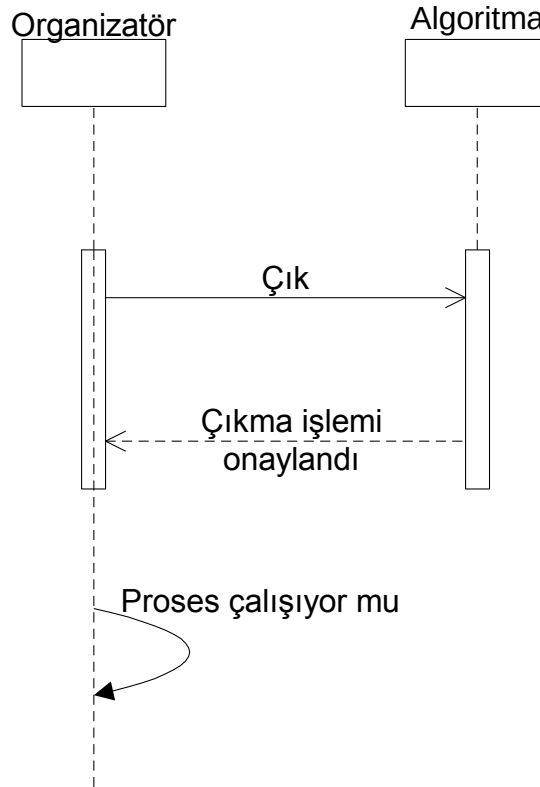
Program çalışması bir şekilde senkronize edilmelidir. Tüm sayısal kontrol sistemlerinde senkronizasyon işlemi zamana göre yapılır. Linux çekirdeğinde olduğu gibi zamanlayıcı kurulacak ve her zaman tıklamasında programın durumu takip edilecektir.

Zamanlayıcının bir görevi de tüm algoritmaların erişebileceği program saatini oluşturmaktır. Linux'un "Jiffies" değerinde olduğu gibi burada da 10 MS de bir yenilenen ve sürekli artan bir program saati sistemi kurulmalıdır. Program saati algoritmalar tarafından erişilebilmesi için paylaşılmış bellekte tutulur ve

algoritmalarla özellikle zaman parametresinin karşılaştırılmasında ve zamana bağlı olan işlerinin yapılmasında kullanılır.

5.9.2 Kilitlenmeler ve çözümler

Kilitlenmeler (deadlocks) de birden çok prosesin birbirlerini beklemeleri sonucunda oluşabilir. Kilitlenmelerin fark edilmeleri oldukça zordur [22], 358. Algoritmalarla dikkat edilir ise “Çık” sinyali gönderilmemesi durumunda algoritma kilitlenir ve sonsuz döngüsünde kalabilir. Bu durumda kalınmaması gerekir. Böyle bir durumun oluşması onaylı görevlendirme ile çözümlenebilir. Bu durumda organizatör ile algoritmalar arasında, aşağıdaki gibi bir sekans diyagramı görünmelidir :



Şekil 5.17 Algoritmaların sonlandırılması

Organizatör tarafından gönderilen “çık” mesajı algoritma tarafından alınır. Bu emrin alındığı organizatöre geri döndürülmelidir. Bu sayede organizatör beklenmedik bir durum olmaması halinde algoritmanın, yani bir Linux prosesinin çıktığını (yani exit() yaptığını) bilir. Ama bunu da şansa bırakmamak için algoritma için yapılan iletişim işlemleri sonrasında daha önceden yaratılma

sonrasında bir yere kayıt edilmiş olan proses ID si kullanılarak o prosesin sistemden kesinlikle çıktığından emin olunabilir.

Algoritmalar Linux prosesleri olduğundan, sistemden çıkmama durumunda sistemde birikirler. Bu birikme sistemde birçok prosesin aynı anda çalışmasına neden olur. Bu durumda sistem aşırı yüklenir. Ayrıca aynı IO değerlerini kullanan algoritmalar birbirlerinin çalışmasını olumsuz yönde etkiler. Böyle bir durumda sistem kaosa sürüklenmiştir. Bu durum sistemi göçürünceye kadar devam eder. Böyle bir durumun uzun sürmesi, endüstriyel prosesi de çok kötü etkiler. Dolayısı ile böyle bir duruma kesinlikle izin verilmemelidir.

5.9.3 Yarış durumları ve çözümler

Özellikle organizatör ve prosesler arasındaki paylaşılan bellekte bir takım yarış durumları oluşabilir. Zira paylaşılan bellekteki değerlerin bazıları okunabilir / yazılabilir. Okuma ve yazma aynı değişken üzerinde farklı zamanlarda yapılabileceğine göre değişkenlerin koruma altına alınması gereklidir. (mutual exclusion.) Bu amaçla Linux işletim sistemindeki semaforlar kullanılır.

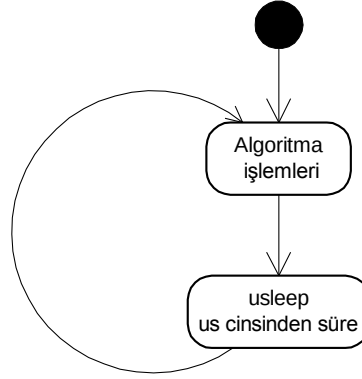
5.9.4 Meşgul döngüler (Busy waiting)

Bu bir senkronizasyon sorunu olmamakla birlikte, CPU kaynağının oldukça kötü kullanılması ve sistem performansının düşmesini beraberinde getirir. Bu durumdan kaçınmak gereklidir. Bu amaç doğrultusunda Linux işletim sisteminde değişik çözümler de sunulmuştur. Örneğin dosya işlemlerinde, özellikle de soketlerin okunmasında “do” döngüsünde dönmek ve bir “handle” ı sürekli okumak yerine, Linux “select” komutu kullanılabilir. Veya zaman gecikmesi için bir “for” döngüsü kullanmak yerine, “sleep” komutu kullanılabilir [22], 360.

Algoritmalar kendi içerisinde bir döngüde dönmek durumundadır. Zira algoritmalar sistemde paralel olarak çalışırlar. Sadece senkronizasyon işlemleri ile bir bütünlük oluşturulabilir. Bu durumda algoritmaların kendi içerisinde bir döngü olacak ve bu döngü için bir meşgul döngü sorunu ortaya çıkacaktır.

Sürekli döngülerde, genel olarak, Linux HZ sabiti gibi bir sabit tanımlanmalı (MS olabilir) ve bu sabitin değeri, sistem gereksinimlerine göre belirlenmelidir. Bu

sabit, algoritma döngüsünde aşağıdaki şekilde görünür. Sistem CPU’yu yormamak amacı ile algoritma döngüsüne “usleep” komutu eklenmiştir. “Usleep” komutu mikrosaniye cinsinden bir bekleme süresi kadar bekler. Bu bekleme esnasında proses, yani algoritma, Linux işletim sistemince “suspend” moduna alınacak ve zamanı gelinceye dek bu durumda bekleyecektir. Dolayısıyla CPU’yu gereksiz yere meşgul etmeyecektir.



Şekil 5.18 Algoritma döngüsünde sürekli beklemenin önlenmesi

5.10 Sonuçlar ve Tartışma

Bu bölümde sistemin tasarımı UML diyagramları ile sunulmuştur. Burada sunulan diyagramların tüm tasarımcılar arasında ortak bir dil ve notasyon kullanılarak yapıldığı anlatıldı. Böylece buradaki verilen sistem dünyanın herhangi bir yerinde programcılar tarafından istenilen programlama dili ile kodlanabilir.

Tasarım genel amaçlı bir yığın kontrolcüsünü oluşturdu. Bu kontrolcüde en düşük düzeyde, algoritmalar bulunmakta ve kullanıcıların kullanabileceği komutlara dönüştürülmektedir. Tanımlı olan komutlar kullanılarak, programlar, yani yığın işlemler oluşturulmakta ve proses kontrolü yapılabilmektedir.

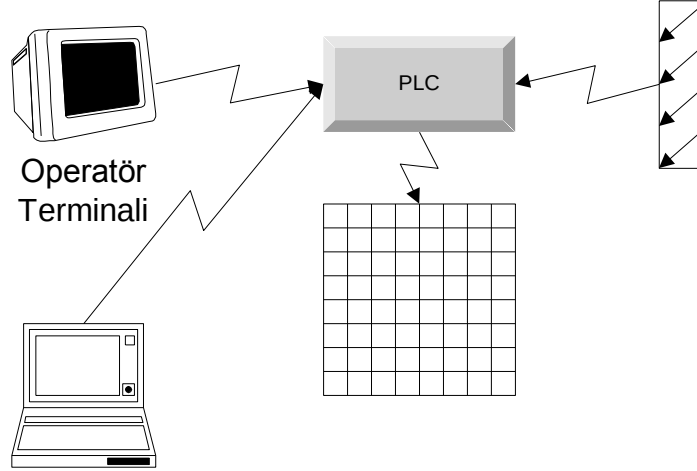
6. SİSTEM YAZILIM MİMARİSİ VE MODULLER

Bu bölüme kadar bahsi geçen tüm bölümler tezin teorik bölümüne ilişkindi. Geçmiş bölümlerde konu ile alakalı bilgiler tüm hatları ile verildi. 6. bölümde, tezi inceleyen araştırmacılar için pratik bir takım yorumlar yapılacaktır. Burada bir önceki bölümde yapılan tasarımlar üzerinden, bir işlemci sanal olarak tasarlanacaktır.

Bu bölümde tasarlanan sanal yığın işlemci için C fonksiyon prototipleri verilecek ve bu prototiplerde yer alması gerekli operasyonlar anlatılacaktır. Araştırmacı buradaki bilgileri kullanarak kendi kodlamasını farklı bir şekilde yapabilir.

Modüler tasarım günümüz yazılım modellemesinin de temelini oluşturmaktadır. Daha önceki bölümlerde de anlatıldığı gibi UML diyagramlarını kullanarak çok iyi bir yalıtım ile, karmaşık sistemler daha küçük ve birbirlerinden bağımsız kümeler şeklinde modellenebilirler [14]. Bu modelleme neticesinde, bahsi geçen her bileşeni modül olarak değerlendirilecektir.

Sistem iki parçalı olarak düşünülecektir. Şekilde tipik bir iki parçalı tasarım bileşenleri görünmektedir. Burada asıl oyuncular Operatör terminali ve PLC'dir. Taşınabilir bilgisayar sistemin devamlılığını ve düzenlenmesini sağlar.



Şekil 6.1 İki parçalı tasarım

- **PLC** : Tüm fiziksel IO kanalları bu birimde bulunur. Daha önce de değinildiği gibi, bu bölüm aptal IO olarak tasarlanacaktır. Aptal IO sisteminde aşağıdaki kanallar bulunur ve bu kanallar belirtilen amaçlar doğrultusunda kullanılabilir .
 - **Analog giriş** : Analog girişler sıcaklık okumalarında, seviye okumalarında, basınç, PH, viskozite vb. değerlerin okunmasında kullanılabilir.
 - **Analog çıkışlar** : Analog çıkışlar motor kontrol devrelerinde, ısıtma sistemlerinde kullanılabilir.
 - **Sayısal giriş** : Sayısal girişler genellikle kullanıcılar ile alakalı olan kontrollerde kullanılır. Bir aktiviteyi başlatmak ve durdurmak gibi işlemler butonlara dolayısı ile sayısal girişlere bağlanır. Bunun dışında bir işlemin sonucu sayısal giriş olarak alınabilir. Bir seviyenin yeterli olup olmadığı, bir motorun sorun yaratıp yaratmadığı vb.
 - **Sayısal çıkış** : Uyarı lambaları, bazı basit motorlar, bazı basit ısıtma sistemleri vb. kontrol işlemleri sayısal çıkışlar ile yapılabilir.

- Operatör Terminali : Bu bölümde gömülü sistem çalışacaktır. Linux, tüm kontrol algoritmaları, yani yığın program, kullanıcı arabirimi vb. programlar burada çalışacaktır.

Böyle bir sistemde eğer herhangi bir birim sorun yaratır ise proses kontrol işlemi tamamen ortadan kalkar. Yani sistem her iki birimin de aynı anda ayakta olması ile mümkündür.

6.1 IO Yönetim Modülü

İki parçalı tasarım, şekil 5.2 'deki gibi bir sistemin tasarlanmasını gerekli kılar. Böyle bir sistemde, seri iletişim kanalı ile PLC birimine IO değerleri gönderilecek ve alınacaktır. Seri iletişimdeki protokol istenildiği şekilde seçilebilir. Burada protokolün varlığı kabul edilecek ve operatör terminalindeki DMA alanının PLC deki alana aktarılabildiği varsayılacaktır.

IO yönetim modülü 2 taraflı bir modüldür. 2 tarafta da farklı incelenmelidir.

6.1.1 PLC

PLC tarafında IO işlemleri daha basit olacaktır. Bu tarafta platform bağımlı kod yazmak gereklidir. Platformların çoğunda, özellikle PLC için uygun platformlarda IO işlemleri kaydediciler vasıtası ile yapılır. Örneğin Hitachi H8 işlemcisinde, ADC çeviricisinin neticesi, ADDR_X adresinden okunur. Benzer şekilde sayısal çıkışlar pinlere, PDX kaydedicilerine bit örgüleri verilerek yazılır. Buradaki fonksiyon prototipleri, aşağıdaki şekilde varsayılabilir :

- **Write_DMA_To_Real_IO(DMA_Base, IO_Register, Count)** : Bu fonksiyon ile “DMA_base” ile verilen adresteki bilginin, IO kaydedicilerine aktarılması düşünülmektedir. “Count”, aktarılabilecek veri miktarını gösterir. “DMA_Base”de ilgili IO tipinin (analog giriş, analog çıkış, sayısal giriş, sayısal çıkış), DMA adresinin bulunduğu adres verilir. Böylece DMA bilgileri ile gerçek IO birebir karşı düşürülebilir.
- **Read_Real_IO_To_DMA(*DMA_Base, IO_Register, Count)** : “Write” işleminin tersine burada “count” ile verilen bilgi miktarı

“DMA_Base”’e okunur. Okumanın kaynağı olarak IO kaydedici adresi verilir.

Görüldüğü gibi gerçek IO’ların devreye alınması basit bir işlemdir. Fakat burada bazı önemli hususlar, konu ile alakalı olmadığından dolayı atlanmıştır. Örnek olarak

- Filtreleme işlemleri : Girişlerin okunur okunmaz geçerli kabul edilmesi pek uygun değildir. Bir takım girişler parazit alabilirler. Bu nedenle okumanın belirli bir süre boyunca aynı değerde veya belirli bir sınır içerisinde kalması ve kendisini ispatlaması gereklidir. Bu durumun çözümü, DMA adreslerinin bir kopyasının oluşturulması ve bu kopya üzerinde bu tip işlemlerin yapılması ile mümkün olabilir. Kopya üzerindeki değerler geçerli sayılabilir ise bu kopya gerçek DMA ya kopyalanır ve iletişim makinesi tarafından işlenmesine olanak verilir.
- Kalibrasyon işlemleri : IO değerlerine bir takım kalibrasyon işlemleri uygulanmak istenebilir. Böyle durumlarda yine kopya bir DMA alanında çeşitli işlemler yapılabilir.
- Özel işlemler : Bazı özel IO tipleri özel bir takım işlemlere tabi tutulmaları gerekebilir. Bu tip IO’lara örnek olarak
 - Sayaçlar : Sayaçlar darbe sayıcılarıdır. Her darbe sayılmalı ve bir önceki değere eklenmelidir. Ayrıca bu değer bir yerde silinmesi gereklidir.
 - Optik mesafe ölçücüler : Ölçülen darbelerdeki faz farkından dolayı yönün belirlenmesi ve sayılan darbelere göre de mesafenin hesaplanması gereklidir.
 - PT 100 sıcaklık okuyucuları : PT100 özel şekilde okunur ve hesaplanır. PT100’lerin değerleri tablolardan okunur. Dolayısı ile bu değerlerin yazılıma gömülmüş olmaları gereklidir.

sayılabilir. Bu tip örneklerde yine özel bir takım işlemlerin okunan değerlere uygulanması ve iletişim makinesine bu şekliyle sunulması gerekir.

6.1.2 Operatör Terminali

IO modülünün diğer tarafında iletişim makinesi bulunmaktadır. Bu makine operatör terminalinde bulunur. Makine, operatör terminali tarafında kullanılan iletişim protokolünün üzerinde veri iletişimini gerçekleştirir. Şekil 5.2’de görünen DMA alanlarının, birbirlerinin aynı olmasını sağlamakla görevlidir.

İletişim makinesini ve PLC’yi birlikte düşündüğümüzde operatör terminalinin istekçi ve PLC’nin ise sunucu olduğu bir sistem söz konusudur. Operatör terminali PLC’den yazma isteğinde bulunacak, bu istek PLC tarafından gerçekleştirilecek; ardından PLC’den girişler okunmak istenecek ve PLC tarafından girişler sunulacaktır.

İletişimin 4 durumlu zaman makinesinin (şekil 5.4) kullanılarak yapıldığını varsayalım. Bu durumda olması muhtemel fonksiyonlar aşağıdaki şekilde düşünülmelidir :

- **Prepare_And_Send_Outputs(DMA_Base, Protocol_Output_Buffer, count) :** Bu fonksiyon, DMA adreslerinde bulunan, algoritmaların vermiş oldukları çıkışları, protokol çıkış tamponuna alır. Protokol çıkış tamponu denilen alan, kullanılan protokole göre oluşturulacak olan paketin bulunduğu alandır. “Count” ise DMA’dan kaç adet verinin transfer edileceğini gösterir. Eğer sistem çıkışlarına ilişkin bilgiler DMA üzerinde bir küme şeklinde biriktirilir ise bu tip bir fonksiyon ile gönderilmesi daha kolay olacaktır. Çıkış paketinin hazırlanmasının ardından “Protocol_Output_Buffer”, seri kanala gönderilir.
- **Receive_Send_Status() :** Gönderilen çıkış bilgilerinin ardından bir alındı paketi gelmesi gereklidir. Alındı paketi, PLC tarafından gönderilecektir. Bu paket ile bir önceki paketin sorunsuz olarak gönderildiği ve bir sonraki duruma geçmemizin mümkün olduğu bildirilir.
- **Prepare_And_Send_Inputs(DMA_Address, Count) :** Bir istek paketi hazırlar ve bu paketi PLC’ye gönderir. Bu paket ile, PLC’deki girişlerin

okunması istenmektedir. Hangi adresten başlayarak kaç adet okunacağı, protokolü kullanarak PLC'ye bildirilir.

- **Receive_Inputs(*DMA_Buffer)** : Gelen pakette, giriş IO bilgileri bulunmaktadır. Bu bilgiler protokol verisinden ayrılarak DMA'daki yerine konulmalıdır.

4 zamanlı makinenin gönderdiği isteklerin yanıtlarını verebilecek olan bir takım rutinlerin, paketlerin alınmasının ardından, PLC tarafından işletilmesi gereklidir. Dolayısı ile yukarıdaki fonksiyonlarının karşılıkları, PLC için de yazılmalıdır.

4 durumlu zaman makinesi sürekli döngüde işletilmelidir. Dikkat edilirse 4 zamanlı durum makinesinde giriş durumu vardır fakat final durumu yoktur. Zira sistem açık olduğu sürece PLC iletişimi de var olmalıdır. İstenirse iletişim makinesi durum geçişleri durdurulabilir.

Döngü Linux işletim sisteminin zamanlayıcısını kullanarak yapılacaktır. Bu zamanlayıcı 10 MS'de tıklayan ve programlanması kolay olan bir zamanlayıcıdır. İletişim makinesi durumlarını takip edebilmek için "jiffies" değerinde olduğu gibi bir durum değişkeni tutulabilir. Bu durumda zamanlayıcı tıklamasındaki kodda, şu şekilde bir makine kurabiliriz :

```
Switch ( DurumDegiskeni )  
  
    Case Send : Prepare_And_Send_Outputs()  
  
    Case SendOK : Receive_Send_Status()  
  
    Case Receive : Prepare_And_Send_Inputs()  
  
    Case ReceiveOK : Receive_Inputs()  
  
    DurumDegiskeni = 0  
  
End switch  
  
DurumDegiskeni = DurumDegiskeni + 1  
  
if ( DurumDegiskeni > ReceiveOK )  
  
    DurumDegiskeni = 0
```

Bu makinede sürekli olarak artan bir DurumDegiskeni vardır. Bu değişken makinenin hangi durumda olduğunu ve hangi operasyonu yapacağını gösterir. Böyle bir değişkenin varlığı, aynı zamanda bir “for” döngüsü kurmayı ortadan kaldırır ve makinenin, zamanlayıcı fonksiyonunun içine gömülmesini sağlar.

IO modülünde iletişim makinesinin yanı sıra algoritmaların IO erişimlerini düzenlemek amacı ile bir takım rutinlerin bulunması uygun olacaktır. Fakat bu fonksiyonlar algoritma başlığı altında açıklanacaklardır.

6.2 Algoritmalar

Algoritmalar şekil 5.7 ‘de görüldüğü gibi 3 ana aşamada incelenir. Bu durumda bir algoritmada şu fonksiyonlar bulunmalıdır :

- **Algorithm_Initialize()** : Algoritmanın ilk değerlerinin verilmesi için çağrılır. Bu çağrı bir kereye mahsus yapılır.
- **Algorithm_Destruct()** : Algoritmanın başlattığı işlemleri geri alabilmesi için, bir kereye mahsus olarak algoritmadan çıkarken işletilir.
- **Do_Algorithm(Algorithm_Class)** : Algoritmanın asıl döngüsüdür. Parametre olarak verilen değer ile dışarıdan gelen mesajlar alınır. Bu mesajlar Şekil 5.7’de “Dış Yönetim İstekleri” olarak belirtilmişlerdir. Şekil 5.13’de hangi tip mesajların gelebileceği sınıflandırılmıştır.

Algoritma Linux prosesi olacağına göre, kendine ait “main” rutini bulunur ve organizatörden ayrı bir program olduğundan, aynı değişkenleri kullanamaz. Bu durumda organizatör ile algoritmalar arasında paylaşılmış bellek bulunması zorunludur.

Algoritma programı aşağıdaki şekilde kurulabilir

Main()

Algorithm_Initialize()

While (GetMessage() != Çık)

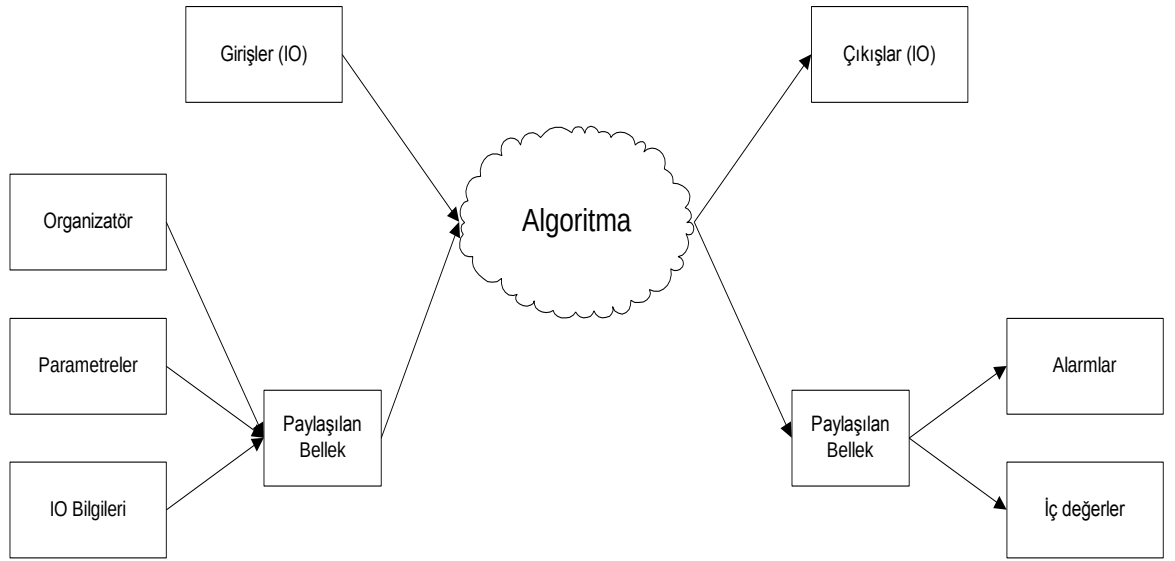
Do_algorithm()

Wend

Algorithm_Destruct()

End Main

Burada Algoritma ile alakalı temel fonksiyonların yanı sıra dışarıdan gelecek ve dışarıya verilecek olan bir takım bilgiler vardır. (şekil 5.8). Bu konuda var olan ilişkiler aşağıda gösterilmiştir :



Şekil 6.2 Algoritma giriş ve çıkışları

Görüldüğü gibi organizatör ve komut bilgileri (parametreler ve IO bilgileri) paylaşılan bellek üzerinden algoritma ile iletişim içerisindedir. IO'lar da paylaşılmış bellekte olmak zorundadır. Çünkü sistemde aynı anda birden fazla algoritma çalışıyor olabilir. Tersine olarak hiçbir algoritma da çalışmıyor olabilir. Program yüklendi durumu gibi. Bu durumda hiçbir algoritmaya IO yönetimi verilemez. IO koordinatörün yönetiminde olmalıdır. Bu durumda algoritmaların IO değerlerine erişebilmesi için IPC sistemi kullanılması gereklidir. Bu amaç doğrultusunda paylaşılan bellek kullanılması uygundur. Fakat şekilde diğer bellek ile karışmaması amacı ile o anlamda gösterilmemiştir.

Algoritmelerde IO'ları düzenleyen fonksiyonlar, iki tane düşünülebilir :

- **Write_IO(DMA_Address, value)** : “Value” parametresi ile verilen bir değeri, paylaştırılmış bellek üzerindeki IO DMA değerine yazar. Fakat bunu yapmadan önce IO paylaştırılmış belleğin orada bulunup bulunmaması, yazılacak değerin uygun ve sınırlar içinde olması gibi kritik şartları kontrol etmelidir. Ayrıca DMA_Address olarak verilen adresin aslında algoritmaya parametre olarak gelen IO değerlerinden biri olması gereklidir. Bu şekilde bir tasarım sayesinde algoritmadaki IO'lar aynı tipten olmak şartı ile, IO kanalı farklı olabilir. Bu durumda algoritmanın çalışması, aynı tipteki her IO için mümkün olup çok esnek bir sistem tasarlanmış olur. Böyle bir tasarımın 2 önemli faydası olabilir :

- Aynı algoritma kodu ile farklı nesneler kontrol edilebilir. Örnek olarak birden fazla kazan, tek bir algoritma ve kazan sayısı kadar komut ile her kazan için ısıtma yapılabilir.
- IO'lardaki sorunlara karşı daha sağlıklı bir yapı kurulmuş olur. Bozulan IO kanalları kolaylıkla sağlam ve boşta olan kanallar ile değiştirilebilir.

IEC 61134 standardında (PLC program yazma standardı. Bu standart ile Instruction List – IL, Structured Text – ST, Function Block Diagram – FBD gibi yazım standartları belirlenmiştir.) bu tip bir özellik yoktur. Projedeki IO'ların değiştirilip projenin tekrar derlenmesi gerekir. Bu işlem ise PC ve sofistike bir eleman gerektirdiğinden, pahalı bir işlemdir. Oysa tasarladığımız sistemde yetenekli bir GUI ile, en basit kullanıcılar dahi bir komutun IO kanalını belirleyebilir.

- **Read_IO(DMA_Address)** : DMA adresi ile verilen bir kanalın değerini geri döndürür. Yazma işleminde olduğu gibi IO paylaştırılmış belleğin orada olup olmasını kontrol etmelidir. Yine yazma işleminde olduğu gibi “DMA_Address” parametresi algoritmaya paylaştırılmış bellek üzerinden gönderilen IO bilgilerinden birisi olmalıdır.

Şekil 6.2 de görüldüğü gibi aslen algoritmanın kendi bileşenleri olan Parametreler, IO bilgileri, Alarmlar ve İç değerler (şekil 5.8) paylaştırılmış bellek üzerinde tutulmaktadırlar. Bu tip bir uygulamanın gerçekleştirilmesi paylaşılmış bellek kullanılarak kolayca gerçekleştirilebilir :

```
Struct Shared_Algorithm_Components  
  
int parameter;  
  
int returned_value;  
  
int alarm;  
  
int io;  
  
end struct
```

Yukarıda örnek bir C yapısı görülmektedir. Bu yapıda bulunan değişkenler bileşenlere örnek teşkil etmektedir. Bu değişkenler daha yaratıcı düşünülerek kendileri içerisinde bir yapı olarak tasarlanabilir ve boyutları da artırılabilir. Burada örnek olması açısından bu şekli ile kullanılacaktır. Böyle bir yapıyı paylaştırılmış bellekten kullanabilmek için aşağıdaki fonksiyonlar düşünülmelidir :

- **Create_Shared_Components()** : Bu fonksiyon ile paylaştırılmış bellekte, yukarıdaki “Shared_Algorithm_Components” yapısı için yer alınır. Daha sonra programdaki herhangi bir kod, bu alanı bağlayarak istediği şekilde kullanabilecektir.
- **Attach_Shared_Components(*Shared_Algorithm_Components)** : Bu fonksiyon ile lokal olarak tanımlanmış bir Shared_Algorithm_Components yapısı paylaştırılmış belleğe bağlanır. Bu amaçla Linux kütüphanesinde bulunan “shmget” ve “shmat” komutları kullanılacaktır. Konu 4.1. “Attach” edilen bir bellek değişkeni, sonrasında normal değişkenlermiş gibi kullanılabilir. Tabi ki paylaştırılmış bellek “pointer” ile gösterilir. Bu durumda örneğin 100 numaralı alarmı vermek için

Shared_Algorithm_Components->alarm = 100; şeklinde kullanılabilir.

- **Shmdt(Shared_Algorithm_Components)** : “Attach” edilen bir paylaştırılmış bellek “detach” de edilmelidir. “Shmdt” fonksiyonu Linux kütüphanesinde bulunur fakat çalıştırılması unutulmamalıdır.

Paylaştırılan değişkenlerin yönetiminde iki tehlikeli durum söz konusu olabilir :

1. **Yarış durumları** : Algoritma bileşenlerinde yarış durumu oluşmaz. Zira aynı anda okunan ve yazılan bir değişken yoktur. Parametreler ve IO bilgileri şekil 6.2 den görüldüğü gibi algoritma tarafından sadece okunur organizatör veya yorumlayıcı tarafından yazılır, benzer şekilde alarmlar ve iç değerler algoritmalar tarafından yazılırken organizatör tarafından okunur. Benzer şekil IO yönetiminde de söz konusudur. Burada IO bölümü ile organizatör arasında hiçbir ilişki yoktur. Organizatör sadece IO bilgilerini PLC’ye gönderir ve alır. Girişler okunduğu çıkışlar yazıldığı müddetçe herhangi bir yarış durumu söz konusu olmayacaktır.
2. **Değişkenin organizatöre iletilip iletilemediği** : Bir alarmın verilmesi örneğini ele alalım. Alarm değişkeni yazılmasına rağmen bu değer bir sebeple organizatöre iletilememesi mümkündür. İkinci ve farklı bir alarmın verilmesi ne şekilde olacaktır ? İkinci alarmın 1. alarm üzerine yazmaması gerekir. Bu nedenle alarm verme sisteminde kuyruk yapısı oluşturulmalıdır. Benzer şekil Linux işletim sisteminin kesme yönetiminde de vardır. Konu 3.2.4.

6.3 Komut Editör Modülü

Komut modülü için yazılması gereken sistem fonksiyonu fazla yoktur. Algoritmalar ile komutlar arasındaki tek fark, algoritmaların arabirimlerinin olması ile birlikte komutların oluşmasıdır. Bu durumda komutların asıl işi arabirimidir.

Algoritma bileşenleri (parametreler, iç değerler vb.) kullanıcı tarafından değiştirilebilir ve yönetilebilir olması isteniyorsa bilgileri kullanıcıdan istenmesi gerekir. Örneğin parametrenin ismi, sınırları vb. gibi. Böyle bir durumda GUI tasarımı söz konusu olacaktır. Eğer kullanıcı tarafından belirlenmesi gerekmiyorsa bu bileşenlerin parametrelerine ihtiyaç yoktur.

Bileşen bilgilerinin kullanıcı tarafından yönetilebilir olmasının istenmesi durumunda, komut modülü ile alakalı olan bir diğer önemli husus bu bilgilerin saklanması gerekliliğidir. Bu doğrultuda text dosyaları veya veritabanının kullanılması mümkün olabilir. Konu 5.6.

6.4 Program Editör Modülü

Program editörü de komut editörü gibi kullanıcıya bağlı bir modüldür. Eğer kullanıcı program yazabilecek ise bu durumda yazılabilecek olan programın kullanıcı arabirimi oluşturulmalıdır. Oluşturulacak olan arabirim ile program oluşturulur ve uygun bir şekilde saklanır. Konu 5.6.

Oluşturulacak olan programda yığın işlemde olması gereken durumların ele alınması gereklidir. Bu durumlar komutların konumlarını ele alır. Komutların seri, paralel, karma yazılması vb.

Aşağıda örnek ve basit anlamda bir program dosyası görülmektedir :

10 P11 P12 P13... IO11 IO12...

4 P21 P22 P23... IO21 IO22 IO23...

4 P31 P32 P33... IO31 IO32 IO33...

6 P41... IO41 IO42 IO43 IO44...

1 P51 P52... IO51...

8 P61... IO61 IO62 IO63...

9

Böyle bir programda, değişik parametrelere sahip komutlar, paralel komutlar, değişik IO'lara sahip komutlar görülmektedir.

1. adımda 10 numaralı algoritmanın varlığı görülmektedir. Değişik parametreler ve değişik IO'lara sahiptir. 2. ve 3. adımlarda aynı algoritmanın değişik parametre ve IO'lar ile kullanılması görülmektedir. Bu tip komutlar örneğin ısıtma işleminin 2 farklı kazana uygulanması olarak düşünülebilir. 1. 2. 3. komutlar seri olarak işlenirler. Yani bir komut bitmeden diğer bir komut başlayamaz. 4. 5. komutlar

birbirlerine paralel yazılmışlardır. Paralel komutlar aynı anda başlarlar ve her ikisi de işini bitirdikleri an alt adıma geçilebilir. Benzer şekilde 6. 7. adımlar birbirlerine paraleldir. 9 numaralı algoritmanın herhangi bir parametresi bulunmamaktadır.

Eğer kullanıcı arabirimi olmayacak ise bu tip bir program herhangi bir editör ile oluşturulmak zorundadır. Oluşturma işlemi için çok iyi hazırlanmış birtakım dokümanlara ihtiyaç bulunmaktadır. Böyle bir programın oluşturulmasının karışık olacağı görülmektedir.

6.5 Yorumlayıcı Modül

Yorumlayıcı modül programı okur ve programa uygun paylaşılmış bellekleri oluşturur. Organizatör modülünü başlatır ve programı yüklendi durumuna geçirir. Yorumlayıcı modül, isminin de belirttiği gibi program dosyasını yorumlar. Bu amaç doğrultusunda aşağıdaki fonksiyonlar düşünülebilir :

- **Read_Command_Line(*Line)** : Programdan bir satır bilgi okur.
- **Create_One_Command(Line)** : Okunan bir satıra göre paylaşılmış belleği oluşturur. Komut parametre ve IO bilgilerini paylaşılmış belleğe atar. Ayrıca geri dönüş ve alarm parametrelerini ilk değerlerine getirilir.
- **Start_Executer()** : Organizatörü başlatır. Organizatör farklı bir program olarak tasarlanabileceği gibi

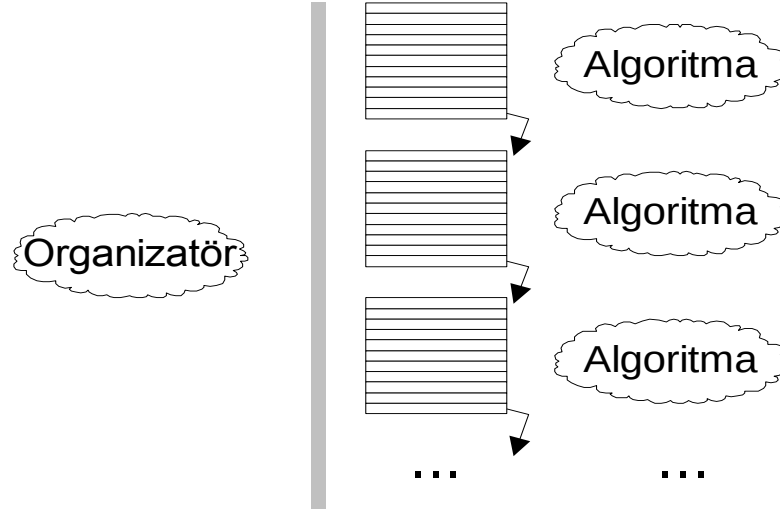
Yorumlayıcı modül ilke olarak tüm programı oluşturur. Böyle bir davranış şüphesiz büyük miktarlarda sistem kaynağı gerektirir. Özellikle çok büyük programlarda bu tip davranış sistemin göçmesine neden olabilir. Bu durumda tasarımın biraz değiştirilmesi gerekebilir.

Yorumlayıcı modül şekil 5.12’de görüldüğü gibi kullanıcı tarafından GUI aracılığı ile tetiklenir. Çok saklı ve fazla bir göreve sahip olmadığından dolayı basit bir modül olarak görülmektedir.

Yorumlayıcıda hata durumları dikkatlice ele alınmalı ve kullanıcıya bir GUI aracılığı ile hatalar ve uyarılar bildirilmelidir. Oluşabilecek hatalar ve dikkate alınması gereken hususlar 5.7. bölümde değinilmiştir.

6.6 Organizatör Modül

Organizatör modül, çalışan programı organize eder. Çalışan programın koordinasyonunu sağlar. Bu amaç doğrultusunda paylaşılan bellek kullanmak zorundadır. Çünkü algoritmalar ve organizatör farklı programlardır.



Şekil 6.3 Organizatör ve algoritmaların ilişkisi

Şekilde organizatör ile algoritmaların ne şekilde haberleşeceği gösterilmiştir. Organizatör algoritmalara paylaştırılmış bellek üzerinden mesajlarını gönderir. Tam tersine olarak algoritmalarındaki alması gereken bilgileri de alır.

Algoritmalarda bahsi geçen ve verinin iletilmemesi sorunu burada da bulunmaktadır. Dolayısı ile tasarımda benzer hususlar dikkate alınmalıdır.

Paylaştırılmış bellekte algoritmaların bileşenlerinin yanı sıra organizatörün mesajları da bulunmaktadır. Bunların ötesinde bilgiler de bulunacaktır. Organizatörün paylaştırılmış belleklere ulaşabilmesi için bir tanesinin adresini bilmesi gereklidir. Diğer belleklere ulaşılması ise her bir paylaştırılmış belleğin birbirini göstermesi şeklinde olacaktır. Bu durumda paylaştırılmış belleklerden bir bağlantılı listenin yapıldığı varsayılabilir. Bu durum şekilde gösterilmiştir. Tüm bu bilgiler paylaştırılmış belleğe yorumlayıcı tarafından yerleştirilmiştir.

Organizatörde aşağıdaki fonksiyonların varlığı düşünülmelidir :

- **Locate_Shared_Memory(Line, Paralel)** : Algoritmanın paylaştırılmış belleğini bulur ve geriye bu belleğin göstericisi döner. Paylaştırılmış bellek, programdaki satır ve bu satırdaki paralel numarasına göre belirlenir.
- **Send_Message(Line, Parallel, Message)** : Programda bulunan bir adıma mesaj göndermek için kullanılır. Öncelikle algoritmaya ait paylaşılmış bellek bulunmalıdır. Sonrasında mesaj, bu belleğe iletilmelidir.

6.7 Sonuçlar ve Tartışma

Bu bölümde tasarıma ilişkin, daha çok yazılıma yakın, bir takım bilgiler sunulmuştur. Bu bilgiler doğrultusunda araştırmacılar kendi yazılım dillerini kullanarak kodlamalarını yapabilirler.

Yazılımın sanal da olsa tasarlanabilmesi, sistemin kağıt üzerinde çalışabildiğini göstermektedir. Bir takım uygulama sorunları çıkacaktır. Zira burada tüm şartların düşünülmüş olduğu söylenemez. Burada sistem sadece ana hatları ile tasarlanmıştır. Ayrıca tasarımın doğru olması halinde bile kodlama hataları bulunabilir.

6.7.1 Gerçek zaman kriterleri

Tez uygulaması, Linux platformunda kullanıcı alanında yapılmıştır. Bu nedenle kesmeler kullanılmamıştır. Kesmeler genellikle cihaz sürücüleri tarafından ihtiyaç duyulmaktadır. Ayrıca bir işletim sisteminin bulunması genellikle aplikasyon geliştiricilerini kesmelerden ve dolayısı ile de donanımdan yalıtmaktadır.

Linux, 10 MS zamanlama kesmesi değiştirilmemiştir. Bu nedenle herhangi bir prosesin 10 MS'den daha kısa aralıklar ile yapacağı işlemler mümkün olmayacaktır. Zaten endüstriyel sistemlerin çoğunluğunda kullanılan pnömatik sistemlerin yanıt verme zamanları 10 MS'nin çok üzerindedir.

Daha fazla gerçek zaman isteği olursa, bunun için değişik metotlar önerilebilir. [7] numaralı kaynakta, var olan Linux sistemi üzerine bir takım yorumlar yapılmaktadır. Bu yorumlarda, Linux'un derinliklerine inilerek, burada bir takım

değişikliklerin yapılmasını öngören çözümler mevcuttur. Bu değişikliklerin bazıları Linux'un davranışını ciddi ölçüde değiştirebilir.

Bazı özel Linux versiyonları kullanılabilir. Bu tip versiyonlarda, Linux davranışları daha gerçek zamanlıdır. Bunun anlamı şudur : İşletim sistemi bir takım zamanlama kriterlerini daha kesin rakamlar ile vermektedir. Tercihen tepki zamanlarının minimum olması istenir. Real Time Linux, QNX Netruino, VxWorks gibi işletim sistemleri gerçek zaman kriterlerini karşılamaktadır. Örneğin VxWorks işletim sistemi, uzaya gönderilen ve ayı incelemesi düşünülen cihazlarda kullanılmıştır.

7. SONUÇLAR

Bir gömülü sistem, işletim sistemi ile birlikte düşünülür ve bu şekilde güçlü bir bütün oluşturur. Bu bütün, işletim sistemi sayesinde donanımdan yalıtılmış bir ortamda aplikasyon geliştirmeyi sağlar. İşletim sistemleri bunların ötesinde aplikasyon geliştiricisine dosya sistemi, ağ yönetimi, proses yönetimi vb. gibi faydalar sağlar.

Yığın kontrol, kendi içerisinde standartlara kavuşan ve birçok prosesin kontrol işleminde kullanılan kontrol mantığıdır. Bu mantık sayesinde seri ve paralel komutlar kullanılarak yığınlar (programlar) oluşturulur ve bu programlar bir yığın kontrolcünde koşarak prosesin kontrol edilmesini sağlarlar.

İşletim sisteminin özelliklerini ve onda kullanılan temelleri örnek alarak bir yığın işlemcisi oluşturulabilir. Fakat işletim sistemlerinden farklı olarak bir yığın işlemcisinde daha farklı bir kontrol ve senkronizasyon sistemi kurulmalıdır. Bu sistem Linux gibi bir işletim sistemi üzerindeki IPC kullanılarak oluşturulabilir.

Bu tezde gerçekten de çok görevli bir işletim sistemi üzerinde, IPC kullanılarak bir yığın işlemcisi tasarlandı. Bu tasarım, kesin bir takım senkronizasyon sınırlamaları ile birleştirilerek, yığın işlem standartlarına uyduruldu.

8. TARTIŞMA

Bu noktada tez ile alakalı olarak tüm bilgiler sunulmuştur. Araştırma esnasında bir yığın işlem yorumlayıcısı oluşturulmuş ve bu işlemci üzerinde kullanıcı özel programların yazılması sağlanmıştır. Araştırmanın bu noktadan daha da ileri götürülmesi mümkündür.

Burada yapılan araştırma endüstriyel uygulamalarda kullanılabilir. Bu amaç doğrultusunda daha da ileri gidilerek sistem, endüstriyel kontrolde çok yaygın olan PLC sistemine yakınlaştırılabilir [16]. PLC'ler endüstriyel kontrolde en çok kullanılan cihazlardır. Bu cihazlar ile bir sistem üzerinde geliştirilmiş olan bir proje bir program halinde PLC cihazı üzerinde koşar. Fakat dikkat edilirse, PLC projeleri kullanıcı tarafından müdahalesi zor olan, projedeki değişiklikler sonrası tekrar derlenmesi gerekli olan projelerdir. Ayrıca GUI ile PLC programı arasındaki bağlantıyı kurmak, eğer OPC kullanılmıyorsa oldukça zor olmaktadır.

PLC sistemlerinde son zamanlarda oldukça önemli değişiklikler olmaktadır. Bu değişiklikler PLC programlarındaki standardizasyon (IEC61131) ve bu konuda geliştirilmiş olan yazılımlardır. Artık bu tip yazılımlar ile PLC özelliği her tür sisteme gömülebilmektedir [17].

İşte bu noktada bu tezde anlatılan yığın işlemci ile PLC sistemlerinin mantığı bütünleştirilebilir. PLC programlarında aynı tip bir fonksiyonun tekrar kullanılması mümkündür ve bu durum “instance creation” kelimeleri ile anlatılır. O fonksiyonun aynısı bir tane daha yaratılır. Bu yaratılan “instance”, kod olarak aynı fakat veri (data) olarak farklı bir alanı kullanır [16].

PLC'lerde var olan program mantığı şu şekildedir : Sistemde yeterli olduğu kadar “instance” projeye dahil edilir. Proje derlenir ve çalıştırılır. Bu aşamada tekrar derleme yapılmaksızın projenin çalışma sistemi değiştirilemez. Örneğin bir

“insance” eklenemez. Fakat program akışı projede düşünülebildiği ölçüde değiştirilebilir. Hangi anda hangi “instance” aktif olacağı belirlenebilir. Kullanıcı burada sadece programın akışına müdahale edebilir. Bu müdahaleyi IO’ları kullanarak yapar [16].

Kullanıcının program yazması ve bu programın bir organizatör olarak PLC projesini yönetmesi mümkün olabilir. Bunun için yığın işlemcide olduğu gibi bir editörün bir de organizatörün var olması gereklidir. Organizatör derlenmiş olan projedeki “instance”ları yönetecek ve kullanıcın yazdığı projeyi proses kontrol manasında PLC projesindeki “instance”ları kullanarak işletecektir.

İşte bu noktada PLC sistemi ile yığın işlemcisi arasındaki farkı değerlendirebiliriz. Buradaki tek fark yapısal anlamda platformda vardır. PLC programı seri olarak işletir. Aynı anda birden çok program koşturmak PLC’lerde mümkün olmamaktadır. PLC sıra ile yaptığı döngüsünde sadece o anda bir “instance”ın aktif olup olmaması gerekli olan şartları inceler eğer şartlar sağlanırsa program akışı o “instance”a doğru değişebilir. Böylece PLC sisteminin (IEC61131) çok basit ve her platforma uygulanabilen bir çözüm olması mümkündür.

Yığın işlemcide ise Linux yani çok görevli bir işletim sistemi mevcuttur. Bu sistem sayesinde aynı anda diğerleri etkilenmemek şartı ile birden çok “instance” paralel olarak koşmaktadır. “Instance”lar dinamik olarak yaratılmakta böylece projenin tekrar derlenmesine gerek kalmamaktadır.

Bu iki sistemin birbiri ile bütünleşmesi mümkün olabilir. Bu bütünleşme birkaç aşamada yapılabilir :

İlk aşamada, PLC sistemindeki proje üzerine yığın işlemcideki editör ve organizatör modellerinin eklenmesi ile yapılabilir. Bu durumda yığın işlemcide bulunan paralel işleme sistemi göz ardı edilmiş olur. Dinamik olarak “instance” yaratmak mümkün olmaz. PLC projesi bir proses ve organizatör 2. proses olarak çalışabilirler. Veri aktarımı ve senkronizasyon paylaşılmış bellek ile yapılır.

İkinci aşamada yığın işlemciye daha yakın bir modele geçilebilir. Bu aşama 1. aşamanın ardından gelmelidir. Bu durumda IEC61131 ile yazılmış olan

“instance”lar yığın işlemede çalıştırılacaktır. Böylece paralel ve dinamik olarak “instance”lar oluşturulabilir ve çalıştırılabilir. Bu aşama çok daha uygun olan bir sistemi beraberinde getirir.

Sistem güvenliği hakkında [7,22,23] derinlemesine araştırmalar yapılmıştır. Güvenlikte dikkat edilmesi gereken noktalar şunlar olabilirler :

- “Denial of Service” atakları : Bilgi güvenliğini tehlikeye atmamak ile birlikte, sistem proseslerini etkiler ve sistemin göçmesine neden olabilirler. Böylece bilginin ortadan kalkmasına neden olurlar.
- Erişim hakları : Erişim haklarının maksimum düzeyde olması birçok soruna neden olabilir. Cihaza giriş yapan ve tam yetkisi olan bir kişi cihazdaki tüm bilgilere erişebilir. Cihaz sahibinin tüm bilgilerine de erişmiş olur. Dosya sistemindeki dosyalara da erişebilir. Bu dosyalara zarar verebilir.
- Ağ güvenilirliği : Ağlar güvenilir ortamlar değildir. Cihazın şifrelerini bilen veya kırabilen biri cihaza girip bir takım bilgilere erişebilir.
- Kod güvenilirliği : Linux sistemleri açık kodlu bir işletim sistemi olduğundan bu kodlar kolaylıkla değiştirilip kötü niyetli kullanılabilir. Bu durumda yazılım sözleşmesinde genellikle sorumluluk kabul edilmediğinden herhangi bir hukuki durum da oluşmaz.
- Uzaktan yönetim : Uzaktan yönetim yapılmak istenen cihaz orada bulunan ve o cihazı taklit edebilen bir sistem sayesinde güvenlik açığı verebilir.
- Tampon taşmaları : Internet sistemlerinde görülen bir tür açıktır. Web sunucu üzerinde yapılan bir istekte tampon taşması yaratılır. Bu kod çalıştırılabilirse sistem her türlü şekilde kontrol edilebilir.

Tüm bu sorunların çözümlerini sisteme yansıtmak gerekir. Zira Linux sisteminin getirmiş olduğu ağ yetenekleri bu sorunları da beraberinde getirmektedir.

KAYNAKLAR

- [1] **Defouw, G., Prat, V.**, 2002. The Matchbox PC: A Small Wearable Platform, IEEE Computer Society.
- [2] **Kamijoh, N., Inoue, T., Olsem, M. C., et al**, 2002. Energy Trade-offs in the IBM Wristwatch Computer.
- [3] **Narayamaswami, C. and et al**, 2002. IBM's Linux Watch: The Challenge of Miniaturization, IEEE Computer Society.
- [4] **Motorola gömülü sistem işlemcileri**,
<http://e-www.motorola.com/webapp/sps/site/taxonomy.jsp?nodeId=01>
- [5] **Intrinsyc**, <http://www.intrinsyc.com/>
- [6] **uCLinux**, <http://www.uclinux.org/>
- [7] **Nakajima, T., Iwasaki, M., Ochiai, S.**, 2002. Issues for Making Linux Predictable, IEEE Computer Society.
- [8] **Wall Kurt**, 2000. Linux Programming by Example. Que, Indiana.
- [9] **I-Linux Documentation**, www.intrinsyc.com/support/ilinux/.
- [10] **Rubini, Alessandro**, 1998. Linux Device Drivers. O'Reilly, USA.
- [11] **Linux**, <http://www.linux.org>
- [12] **RedHat Linux**, <http://www.redhat.com>
- [13] **Hekman, Jessica P.**, 1997. Linux in a Nutshell. O'Reilly, USA.
- [14] **Suzuki, J. and Yamamoto, Y.**, 2002. Toward the interoperable Software Design Models : Quartet of UML, XML, DOM and CORBA, IEEE Computer Society.
- [15] **Karlsson, J.S., Lal, A., Leung, C., Pham, T.**, 2002. IBM DB2 Everyplace : A small footprint Relational Database System, IEEE Computer Society

- [16] **Dr. Wolfgang Brendel – (Infoteam Manager)**, 2002. Kişisel görüşme.
- [17] **Infoteam**, <http://www.infoteam.de/>
- [18] **Eliar**, <http://www.eliar.com.tr/>
- [19] **Gattliff William**, The Linux Kernel's Interrupt Controller API
- [20] **Gattliff William**, The Linux 2.4 Kernel's Startup Procedure
- [21] **Curry David A.**, 1996. Unix Systems Programming. O'Reilly, USA.
- [22] **Goerzen John**, 2000. Linux Programming Bible. IDG Books, CA
- [23] **Stajano Frank, Isozaki hiroshi**, 2002. Security Issues for Internet Appliances. , IEEE Computer Society

ÖZGEÇMİŞ

İbrahim Niyazi Ülgür, 21.01.1976 tarihinde Almanya Köln şehrinde doğmuştur. Anne ve babası İzmir ilinde doğmuş büyümüş ve evlenmişlerdir. 5 kardeşten ikinci büyük olanıdır. Halen bekar olarak yaşamaktadır.

İlköğretimini İzmir ilinde yaptıktan sonra yüksek eğitimi almak üzere İstanbul'a gelmiştir. İzmir'de İzmir Çınarlı Teknik Lisesi ile başlayan bilgisayar eğitimi, yüksek eğitiminde yine bilgisayar üzerine devam etmiş ve 1994 yılında başladığı İstanbul Teknik Üniversitesi Kontrol ve Bilgisayar Mühendisliği programından, 1999 yılında mezun olmuştur. 1999 yılında yine İstanbul Teknik Üniversitesinde Bilgisayar Mühendisliği Bölümünde yüksek lisans eğitimine başlamıştır.

1999 yılında üniversiteyi bitirmesinden sonra iş hayatına atılan İbrahim Ülgür, İstanbul Teknik Üniversitesinde Araştırma Görevlisi olarak işe başlamıştır. 1 yıllık memuriyet hizmetinden sonra 2000 yılında Eliar Elektronik A.Ş.'nde yazılım mühendisi olarak ikinci işine başlamıştır. Halen Eliar Elektronik A.Ş.'nde proje yöneticisi olarak devam etmektedir.

Tenis oynamak, bisiklet kullanmak yapmayı sevdiği spor aktiviteleridir. Sosyal etkinlik olarak dil öğrenimini sevmektedir. İngilizce yi iyi bilmekte, Almanca'yı ise orta seviyede kullanabilmektedir.

İletişim bilgileri :

Cep Telefonu 1 : +90 542 584 79 53

Cep Telefonu 2 : +90 505 524 63 06

Cep Telefonu 3 : +90 555 257 65 70

E-Mail: ulgur@hotmail.com