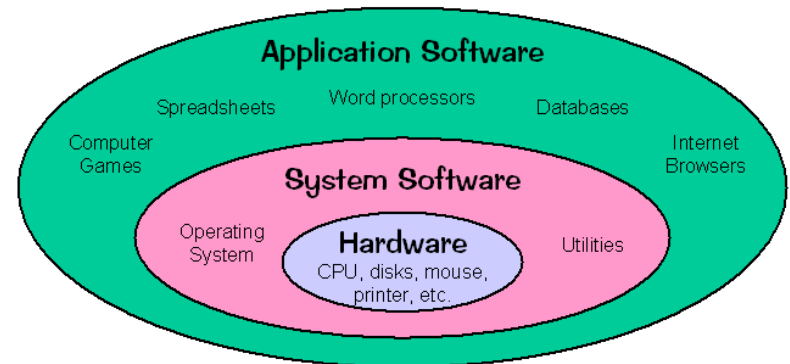


What is System programming?

- ❑ System programming : **is the use of system tools for program development.**
- ❑ In addition to programming information, it requires knowledge of computer architecture and operating system.
- ❑ While services are provided directly to the user through application programs, services are provided to application programs through system programs.
- ❑ In system programming, programs that interact with operating system services are written.

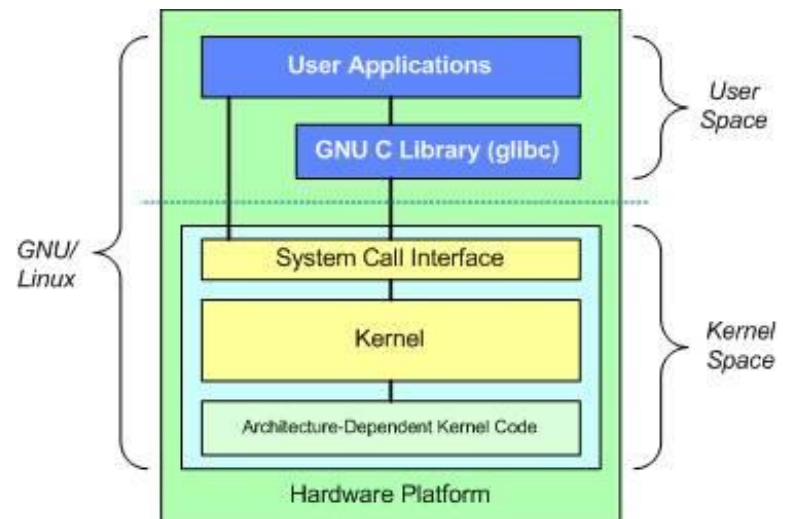
System programming

- Computer; It can be divided into three main parts: hardware, system software and application software.
- It is used to run hardware and system software application programs.
- System software creates an abstraction layer between hardware and application software.



System programming

- ❑ With the abstraction layer created by system libraries, we can use a function without knowing the details of the hardware.
- ❑ For example, the application can be used on that system as long as a system includes similar libraries, such as the GNU C library seen in the Linux architecture provided alongside.
- ❑ The use of system tools enables the creation of standards so that the developed programs are easily transferred to other computers.



Kaynak: <http://www.ibm.com/developerworks/library/l-linux-kernel/>

Why C in system programming

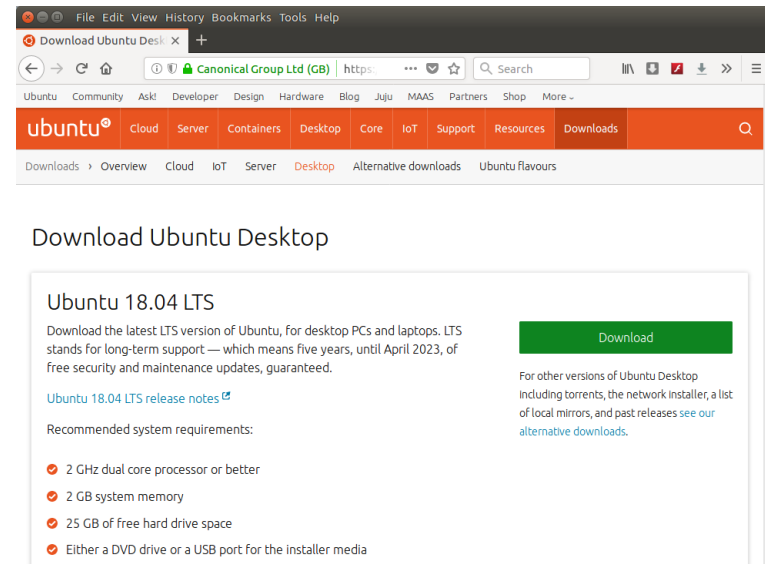
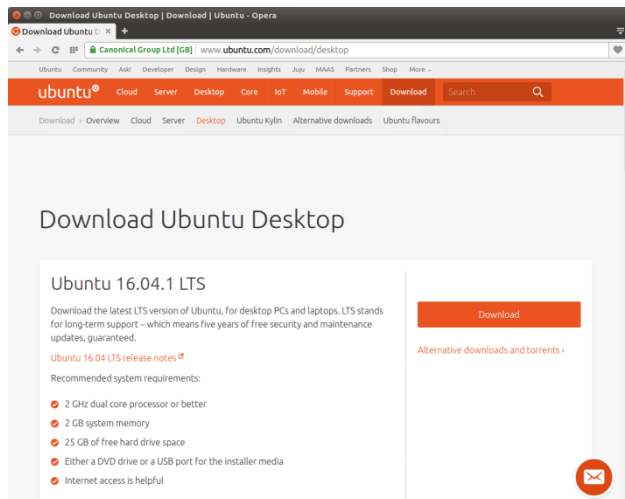
- Reasons for using C in system programming:
C program; It is the basis of the modern computer by using in applications such as operating systems, device drivers, network servers.
- C programming has the least abstraction. Therefore, it is closer to the hardware.
- Many C statements can be converted directly to machine code.
- In C programming, memory can be accessed via pointers, thus enabling access to parts of the system.

Why Linux in system programming?

- Because it is open source, Linux based operating system is preferred in system programming lesson.
- Details about the operation of the Linux-based operating system can be examined. Parts can be added or replaced.
- It is not a closed system because it is not commercial.

Operating system

- You can use the Linux-based Ubuntu operating system to perform the course-related applications.



<https://www.ubuntu.com/download/desktop>

Main topics

The main topics of the course can be listed as follows:

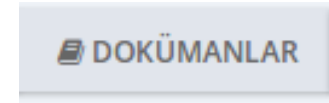
- Basic information about C programming language
- C Pointers, malloc, strings, etc...
- Files and directory files, Signals
- Links (links), Shell forwarding, Shell Script
- Reading / writing from file (File I / O), system calls and buffer usage
- System calls and in / out
- Symbolic language (Assembly) (local variables, functions, branching)
- Processes (processes) and related system calls (fork, exec, dup, pipe)
- Interprocess communication, signals

Evaluation

Çalışma Tipi	Oran
1. Ara Sınav	%75
1. Proje / Tasarım	%10
1. Performans Görevi (Uygulama)	%10
2. Performans Görevi (Uygulama)	%5
1. Final	%50

References

- Sabis course documents



- http://web.eecs.utk.edu/~plank/plank/classes/cs360/lecture_notes.html
- http://web.eecs.utk.edu/~huangj/cs360/lecture_notes.html

CS360 -- Lecture Notes

web.eecs.utk.edu/~jplank/

CS360 -- Systems Programming

James S. Plank --- Spring, 2019

Lecture Notes

Although you can read the notes here, I have put them on bitbucket, and you can grab them to compile and use on your own machine. I strongly suggest that you do so, because hands-on work is how you learn. To grab the lecture notes so that you have all of the programs on your own machine, do:

```
UNIX> git clone https://jimplank@bitbucket.org/jimplank/cs360-lecture-notes.git
```

The lecture notes will be in the directory `cs360-lecture-notes`. You should occasionally do a "git pull" in that directory, to make sure that your notes are up to date.

Lecture notes on bitbucket

- [Lecture 1](#): "C Stuff 1:" Getting Started with C, Scalar Types and Aggregate Types
- [Lecture 2](#): "C Stuff 2:" Pointers, Casting, Malloc, Segmentation Violations and Bus Errors
- [Lecture 3](#): Pointer Arithmetic (Small Lecture)
- [Lecture 4](#): Strings in C
- [Lecture 5a](#): Libfdr -- The Code
- [Lecture 5b](#): Libfdr -- Fields
- [Lecture 5c](#): Libfdr -- Jvals
- [Lecture 5d](#): Libfdr -- Dllist: Doubly-Linked Lists
- [Lecture 5e](#): Libfdr -- JRB: Red-Black Trees
- [Lecture 6](#): [Some Basic Terminology](#)
- [Lecture 7](#): [Introduction to System Calls and I/O](#)
- [Lecture 8](#): [Cat and Buffering](#)
- [Lecture 9](#): [Links](#)
- [Lecture 10](#): [Sh Redirection](#)
- [Lecture 11](#): [Stat and Opendir/Readdir/Closedir](#)
- [Lecture 12](#): [Prsize -- recursive directory traversal](#)
- [Lecture 13](#): [Umask and Other System Calls](#)

CS360 -- Lecture Notes

web.eecs.utk.edu/~huangj/

CS360 -- Systems Programming

Jian Huang --- Fall 2019

Lecture Notes

All Lecture Notes

- [Some Fundamentals](#), some of the things that I assume as prereq's (taken from my CS302 notes)
- [Stuff about C](#)
- [Libfdr lecture notes](#)
- [Red-Black Trees](#)
- [Error Checking](#)
- [Chapter 1](#)
- [Introduction to System Calls and I/O](#)
- [Cat/Buffering](#)
- [Echo](#)
- [Links](#)
- [Stat](#)
- [Prsize -- recursive directory traversal](#)
- [Logfile -- atomic actions](#)
- [The Setuid Bit](#)
- [Taking Time](#)
- [Assembler 1](#)
- [Assembler 2](#)
- [Printing out the stack](#)
- [Assembler 3](#)
- [Setjmp/longjmp](#)
- [Memory](#)
- [Malloc 1](#)
- [Malloc 2](#)