



Foundations to Computer System Design
Professor V. Kamakoti
Department of Computer Science and Engineering
Indian Institute of Technology, Madras
Module 12.4
Understand the Operating System - Compiler Interactions

(Refer Slide Time: 00:17)




Module 12.4

Understanding OS Interface



Welcome to module 12.4 and now we will start understanding the OS interface in a great detail, now we know we have written the compiler now you know how to translate a jack file to a VM file, we know how to translate a VM file to a ASM and then assemble ASM to this and then...so we are quite now familiar with all these things right, so now let us see before we proceed further we will now understand what is the OS interface for a given program.


(Refer Slide Time: 0:57)

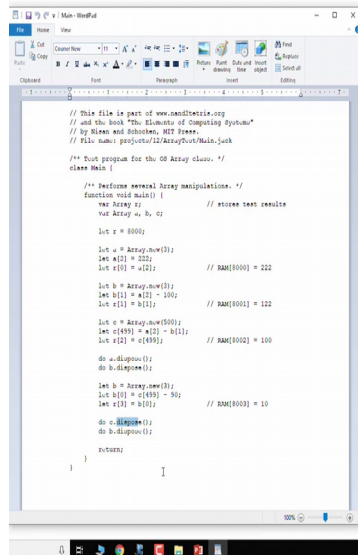
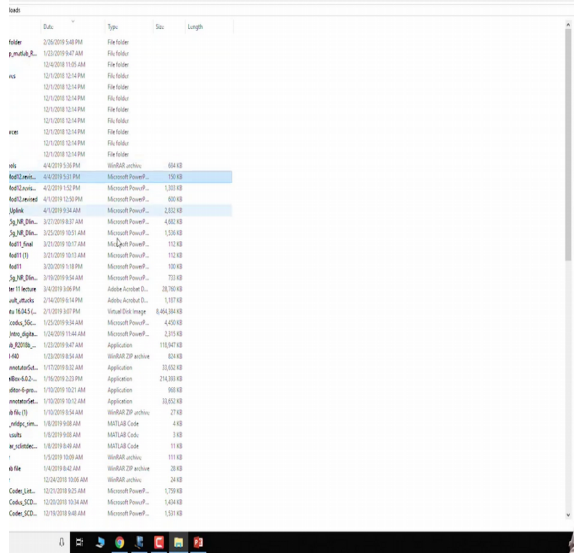


The OS

- Go to tools/OS directory
- We see several *.vm files
- Array.vm, Keyboard.vm, Math.vm, Memory.vm, Output.vm, Screen.vm, String.vm, Sys.vm
- Study these in detail

Handwritten notes:
- Project tools/OS
- User Code calls to OS routines
- Main Jack OS





The OS

Project for Java / JS

*User Code calls to OS services
Manipulate OS*

- Go to **tools/OS** directory
- We see several *.vm files
- **Array.vm, Keyboard.vm, Math.vm, Memory.vm, Output.vm, Screen.vm, String.vm, Sys.vm**
- Study these in detail



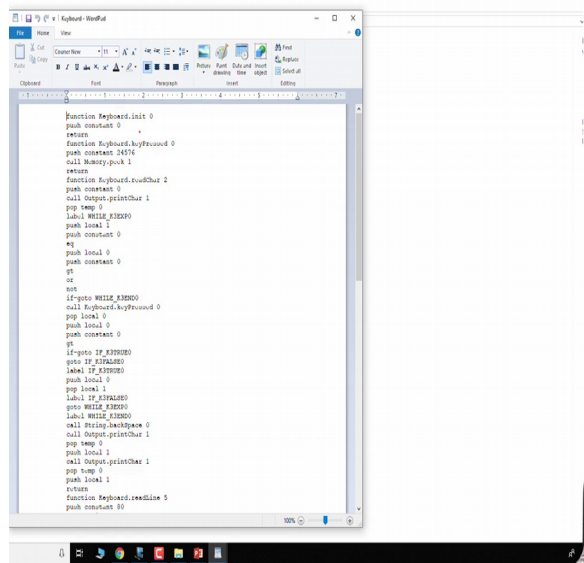
We have seen so many programs right, so if you actually look at some of the programs that you would like to experiment here let us go to the tools directory here, let us take one such example, so I am just taking the example of one of the programs jack programs here, so let us say... Now if you look at this jack program this is called array.test this is available in the projects 12 directory so this is available in your nand to tetris project 12 array test main.jack right. Once you look at this program there are lot of routines that this is basically taking for example array.new similarly array.new here then some disposal teams of c and b where a, b and c are arrays so this is equal enter array.dispose et cetera.

So essentially that means these array functions as you see here or methods that you are seeing here, we are actually basically using something that is available with the operating system okay, so if I say array.new, so this is something that is available with the operating system, so the array all this array functions we assume somebody else's is giving us and who is giving us the operating system is giving us. Similarly if you go back to C programming language you will have print abs scan abs all these things, these are given by the operating system that is why you include STDIO.H and inside STDIO.H you actually see these functions there, so now to go back to where we are every program basically has 2 components, component number 1 is it has whatever your writing it as something that the original user code and there are calls to operating systems routines.

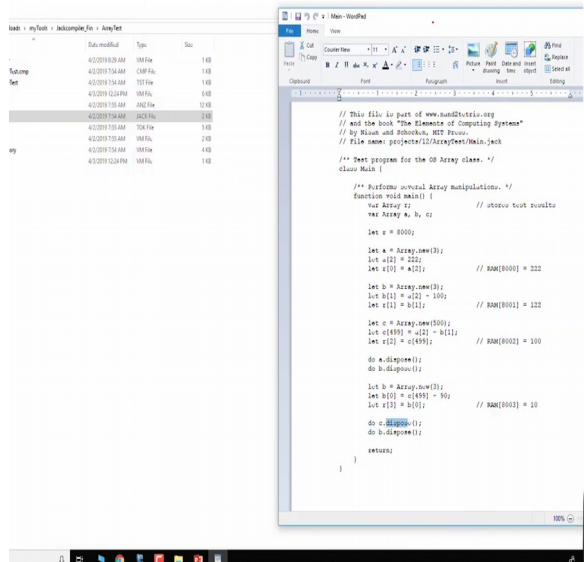


The same thing is what we are seeing here, so we are now having a lot of OS functionalities that are available and they basically use these functionalities these are provided to us as VM files, so you write say main.jack as I have shown, you compiled this you get the .vm files. These .vm files will have certain calls for keyword or math functions or arrays and these are already provided to you by the operating system okay. Now that you find these files, so these files are available in that tool/OS directory so in nand to tetris if you have project directory and of course you would have the tool directory.

So in the tools.OS directory these files are there, so I will just show you the tools.OS directory as we see here, so if you just go back to this, this is the OS directory. Now if you just see array, now you see array.new here and then you say array.dispose here, so 2 functions are part of this array.new and array.dispose similarly if you go to math for example math.init and then you have several other functions so large large files so you have several functions inside math okay. So if you just scan through this entire thing you will find many functions here right for example math.max right so like that.

(Refer Slide Time: 5:37)



```
function Keyboard::init() {
    push constant 0
    return
}
function Keyboard::keyPressed() {
    push constant 24576
    call Memory::push 1
    return
}
function Keyboard::readChar() {
    push constant 0
    call Output::printChar 1
    pop temp 0
    label WHILE_KB_PRESSED
    push local 1
    push constant 0
    eq
    push local 0
    push constant 0
    gt
    or
    if-goto WHILE_KB_PRESSED
    call Keyboard::keyPressed 0
    pop local 0
    push local 0
    push constant 0
    gt
    if-goto IF_KB_PRESSED
    label IF_KB_PRESSED
    push local 0
    push constant 0
    eq
    pop local 1
    label WHILE_KB_PRESSED
    goto WHILE_KB_PRESSED
    label IF_KB_PRESSED
    call String::charAt 0
    call Output::printChar 1
    pop temp 0
    call Output::printChar 1
    pop temp 0
    push local 1
    push constant 1
    return
}
function Keyboard::readLine() {
    push constant 80
    return
}
```



```
class Main {
public:
    Main() {}
    ~Main() {}
};

int main() {
    // This file is part of www.nand2tetris.org
    // and the book "The Elements of Computing Systems"
    // by Nisan and Schocken, MIT Press.
    // File name: project02/ArrayMain.cpp

    /** Test program for the OS Array class. */
    class Main {
    public:
        /** Performs several Array manipulations. */
        function void main() {
            var Array a; // storage test pointer
            var Array b, c;

            let x = 8000;
            let a = Array.new(3);
            let a[0] = 222;
            let a[1] = 4121; // RAM[8000] = 222

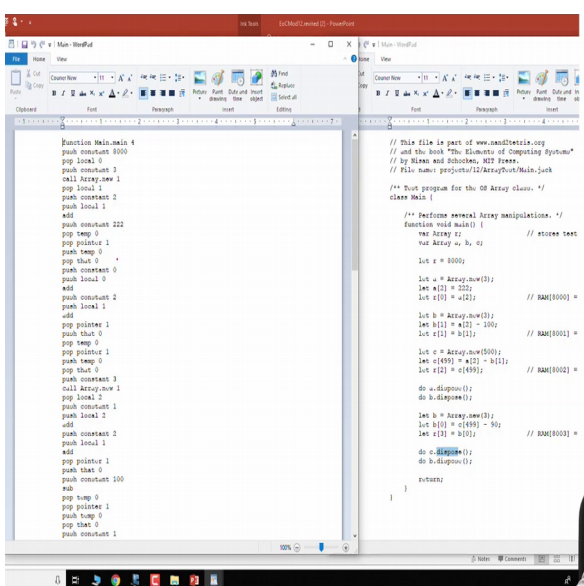


            let b = Array.new(3);
            let b[0] = a[0] - 100;
            let b[1] = a[1]; // RAM[8001] = 222

            let c = Array.new(500);
            let c[499] = a[2] - b[1];
            let c[2] = c[499]; // RAM[8002] = 100



            do a.dispose();
            do b.dispose();

            let b = Array.new(3);
            let b[0] = c[499] - 50;
            let c[3] = b[0]; // RAM[8003] = 10

            return;
        }
    };
}
```



```
function Main::main() {
    push constant 8000
    pop local 0
    push constant 1
    call Array::new 1
    push local 1
    push constant 2
    push local 1
    add
    push constant 222
    pop temp 0
    pop pointer 1
    push temp 0
    pop that 0
    push constant 0
    add
    push constant 2
    push local 1
    add
    pop pointer 1
    push that 0
    pop temp 0
    pop pointer 1
    push temp 0
    push that 0
    push constant 1
    call Array::new 1
    pop local 2
    push constant 1
    add
    push constant 2
    push local 1
    add
    pop pointer 1
    push that 0
    push constant 100
    mul
    pop temp 0
    pop pointer 1
    push temp 0
    pop that 0
    push constant 1
    return
}
```



Similarly your keyboard, so keyboard.init, keyboard.readcare, keyboard.keypressed okay all these things are there. Now you are free to use these as a part of your jack file and automatically when your compiler will basically call these functions right it will just say call of this function, so when you go back to the way... Let us see this example when you look at this is the array. This is the jack file that you are seeing if you see the vm file for that if you see the same vm file for this you will see that this is the vm file and this has basically called array.new.

So you are saying array.new here call array.new one and where is this array.new? This is there in array.vm file right so there are many routines that you can write as a part of your jack code in the compiler will automatically take those code from these vm files this is how you are writing your printer, scanner statements in C and finally you are not coding print f and scanner, somebody else is coding printf and scanner, how is that? The operating system and so that is why you include STDIO.H right and in the STDIO.H these prototypes for all this printer, scanner are there and that is how they are. So somewhere we need to basically link your jack file with these vm files, so that is what we called as (7:28) okay.

(Refer Slide Time: 7:32)

The Interface

- `let length = Keyboard.readInt("How Many Numbers");`
- String is created and given to `Keyboard.readInt` as a argument (Array argument).
- Check out that `Keyboard.readInt` will deallocate this memory after use. If you open `Keyboard.vm` and look at the function `Keyboard.readInt` – it has a

`call String.dispose` statement.
- It will put the integer read on TOS which will be popped into variable length

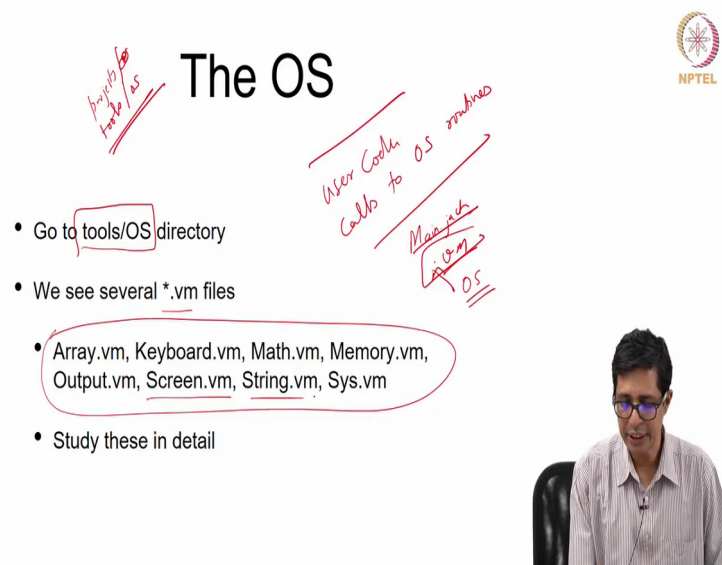
Now we will go ahead and see something, before we go into linking we also need at the compiler stage we also need to understand certain stuffs for example let us take this jack command, `let length is equal to keyboard.readInt how many numbers?` Right so this will basically land us up with `call keyboard.readInt` with one parameter, what is this parameter? This is going to be string constant, so this parameter will be a memory location say 1000, so this will be a memory location which is 500 for example, on 500 what you will see, you will

see a link to 1000, so from 1000 onwards 1000, 1001 like this we will have H, O, W, blank, M like that, so this will basically give this call keyboard.readInt, the parameter that should be before you call keyboard.readInt the parameter that should be at the top of the stack should be 500 sorry the parameter that you see on the top of the string should be 1000.

So what it will do the moment this call keyboard.readInt is called with the parameter 1 on the top of the stack it will be 1000, so immediately this will go to 1000 and start reading that string till it finds a null, so this will lead the string and then that will be process a parameter, so the keyboard.readInt is going to read this string, so what is given as an input to keyboard.readInt is the starting address of the string okay, so in the real compiler if you see this would have been a string constant and how do you process this spring constant?

We just create string length number of locations using string.new right and then we append that will return a 1 null string, so that will return string.new would have returned 1000 with some 20 phases let us say this is some 18 characters, 10 characters or 12 characters, so 12 characters here and it will be initialise to null but (())(10:14) research pace for 12 characters. Now you start appending character by character H then O then W and all, so this particular string constant is actually a term, this term will basically return you on the top of this pack this 1000 and that 1000 is given as an input argument to this and that is how this keyboard.readInt will start reading it.

(Refer Slide Time: 10:50)





Project for Java OS

The OS

User Code calls to OS runtime

Main.java OS

- Go to `tools/OS` directory
- We see several *.vm files
- `Array.vm`, `Keyboard.vm`, `Math.vm`, `Memory.vm`, `Output.vm`, `Screen.vm`, `String.vm`, `Sys.vm`
- Study these in detail



program that memory should be released back otherwise you will land up with what we call as the memory leak. Please remember this term whenever you study compilers or operating system especially compilers you may get a much more deeper understanding of memory leak, so what is memory leak? If I basically create some memory for my program, in this case I have created a memory of say 20 locations for storing this how many numbers, this should be released back to operating system.

(Refer Slide Time: 12:58)

The screenshot shows a Notepad window with the following assembly code:

```

function String.new 0
push constant 1
call Memory.alloc 1
pop pointer 0
push argument 0
push constant 0
push constant 14
if-goto IF_STRING0
goto IF_STRING0
label IF_STRING0
push constant 14
call Keyboard.readInt
pop temp 0
label IF_STRING0
push argument 0
push constant 0
if-goto IF_STRING1
goto IF_STRING1
label IF_STRING1
push argument 0
call Array.new 1
pop this 1
label IF_STRING1
push argument 0
push constant 0
pop this 0
push constant 0
pop this 2
push pointer 0
return
function String.dispose 0
push argument 0
pop pointer 0
push this 0
push constant 0
if-goto IF_STRING2
goto IF_STRING2
label IF_STRING2
push this 1
call Array.dispose 1
pop temp 0
label IF_STRING2
push pointer 0

```

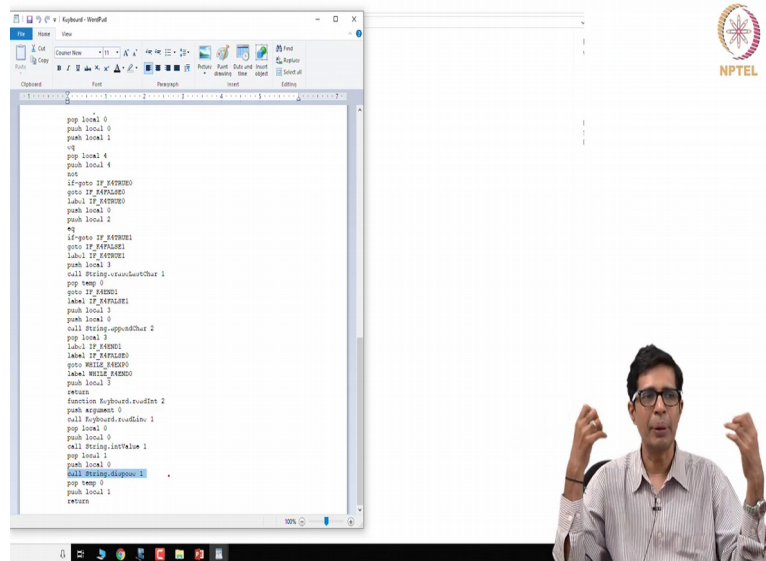
The NPTEL logo is visible in the top right corner.

The screenshot shows a PowerPoint slide titled "The Interface" with the following content:

- let length = Keyboard.readInt("How Many Numbers");
- String is created and given to Keyboard.readInt as a argument (Array argument).
- Check out that Keyboard.readInt will deallocate this memory after use. If you open Keyboard.vm and look at the function Keyboard.readInt – it has a `String.dispose` statement.
- It will put the integer read on TOS which will be popped into variable length

Handwritten notes in red ink include "Memory leak" and "Call Keyboard.readInt()". A diagram shows a stack with elements 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20. The top element is 1, and the bottom element is 20. A red circle highlights the top element 1, and a red arrow points to it with the text "Call Keyboard.readInt()". Another red circle highlights the bottom element 20, and a red arrow points to it with the text "String.dispose".

The NPTEL logo is visible in the top right corner.



So what does `string.new` do basically if you go back to `string.new` here in the code, so this is on the OS let us go to `string` and let us see `string.new`, it actually allocates call memory alloc, so it actually allocates that many memory for storing the string, so it actually causes the operating system. `String` itself is an opening system where it calls the memory management part of the opening system and says give me some memory locations, so the operating system per se will maintain something called the list of freely available memory from that it will take some memory and give it this `memory.alloc` will give it to `string.new` saying this is the total number of locations I have from this you take this much and you use this and this is from the starting address.

So it is the responsibility of the program to return it back to the memory because later somebody else can use that memory or you yourself can use it at a later point of time, so any memory that you request from the operating system has to be disposed of right so in this case, so that is something that you need to understand, so whenever in this case when you say `keyboard.readInt` you are creating some memory locations and giving it to `keyboard.readInt`. Now please understand who is going to return back to the memory?

So that is where you need to understand the routines in very great details, now if you see that this `keyboard.readInt` basically reads this string and after reading this entire string it goes on displays on the screen, if you remember it will say how many numbers and then this `keyboard.readInt` itself will dispose it off, so your program which uses ... This is your program let `length equal to keyboard.readInt` is a statement in your program, your program creates that space for that how many numbers and it calls `keyboard.readInt`, the `keyboard.readInt` actually disposes the memory right, so if you go back to this `keyboard`, if

you look at keyboard.readInt okay let us quickly search for that yes so the keyboard.readInt file if you see that you see that it disposes the string right, so this is how when you call OS routines you should understand what the OS routines is doing specifically from the memory handling point of view otherwise what will happen is you will land up with memory leaks.

Memory leaks is that you have taken the memory and before you finish you have not written by the memory, so the operating system will not know that the memory is available, so that memory you do not use it but you not returning it back then unnecessarily this memory will be hanging without nobody to use it right and that is called memory leak and we have to be very careful not to land up with memory leak and that is one very important aspect of compilation that you will study in a full-fledged compiler course but I am just giving you the thought process of how this can happen right. Now of course after this keyboard.readInt is completed, it actually reads integers from the keyboard and it will put the integer right on top of the stack which will be popped onto this variable length, so that this length assignment gets over, so this is something that you need to understand as a part of this.

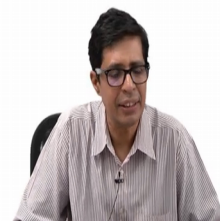
(Refer Slide Time: 16:56)

Linking



- You have a set of jack files – Main.jack; Bat.jack; Ball.jack
- Compile all to get Main.vm; Bat.vm; Ball.vm using your compiler
- Run it on the VMEulator available in tools/.
- Put Main.vm, Bat.vm, Ball.vm and all VMs in the OS directory into a single file. This is called STATIC LINKING
- Compile it using VM to Assembly translator
- Assemble it using Assembly to HACK assembler
- You will find executable too large to execute
- Hence notion of Dynamic Linking

*Program Executable
VM Emulator*



The OS

*Project 12
tools OS*

*User Code
calls to OS
Main Jack
OS*

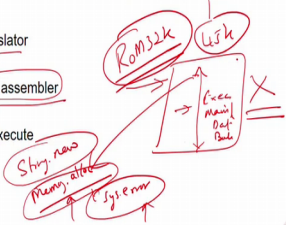
- Go to tools/OS directory
- We see several *.vm files
- Array.vm, Keyboard.vm, Math.vm, Memory.vm, Output.vm, Screen.vm, String.vm, Sys.vm
- Study these in detail



Linking

*Interest of
Memory fragments
Program Executor
VM emulater*

- You have a set of jack files – Main.jack; Bat.jack; Ball.jack
- Compile all to get Main.vm; Bat.vm; Ball.vm using your compiler
- Run it on the VMEmulator available in tools/.
- Put Main.vm, Bat.vm, Ball.vm and all VMs in the OS directory into a single file. This is called STATIC LINKING
- Compile it using VM to Assembly translator
- Assemble it using Assembly to HACK assembler
- You will find executable too large to execute
- Hence notion of Dynamic Linking



Module 12.5

Project 12



Okay now what has happened? Now let us go back, now you have a set of files say for example Main.jack, Bat.jack, Ball.jack let us say there are 3 jack files we compile all of them to get each one of them to get Main.vm, Bad.vm, Ball.vm with this. Now substance foremost these main bat and ball can also use some of the OS routines, if you use the VM emulator that is available in your tools directly, you go and click on the directory that actually stores this 3 dot vm file it will load all the vm files and it will assume it has inbuilt OS functionalities. The vm emulator has all the OS functionalities are inbuilt so you need not put it as part of your vm files, so you can execute this and see whether your program is executing correctly. Program execution can be first tested on the VM emulator.

So as first part of your project 12 you go and see in the project 12 directory you will have many such...if you go back to the project part of your directory, so project 11 of your directory you see so many files on the project 11 average, complex arrays so many directories. Go to each one of these directories compiles all the vm files there, so you compile main file using the compiler, so if you go to the complex array you have Main.jack so compile this were to convert to bin again you have Main.jack similarly go to Pong you have bad ball main pong game compare all the vm files then similarly 7 we will get this and then we can also see the square again you have Main square game.

So all this 6 directories the files you can compile basically now use your real emulator first to check whether they are working correctly right. Now that is the 1st step, so that is what we want you to do as a part of project 11 right and now that you have finished the compiler now you can compile all these files and see whether they are actually executing properly on the vm emulator right so that is something that you need to check up here okay. Now when that is done, so this is over and now your program is working correctly, now you want to execute it on the finally where you have to execute it on the hardware simulator.

So what you do is you put Main.vm, Bat.vm, Ball.vm everything together as a single file and since they are all using all the VM in the memory take all the VM is in the OS directory into a single file right for example you will have array.vm, keyboard.vm, math.vm, memory.vm all these things you put into a single file and then you create one massive executable file and that is basically called exec.vm that is an executable file, so this is called static (())(20:40). Now you can compile it using vm to assembly translator it will get translated to assembly then you can assemble it using assembly to HACK assembler.

So finally you will get the .hack file which will have the entire program. This is basically what an...now you will find that for your machine that we have ROM32k. This is where all your instructions should be stored, if I take all the vm files in the OS directory that itself will come to 45K essentially this executable that you are creating cannot execute on the ROM32K, it is in this machine, the hack machine that you have created this whole thing cannot execute because it will exceed the size of whatever you have as 32K, so that is the most important thing that we need to keep in mind right, so now what I have done at the time of compilation itself I have put all the main, bat, ball and all the vm in the OS directory together as a single file.

This is done at the time of compilation not at the time of execution that is why this is called static linking right. Now by doing this static linking what did we land up, we have landed up with an executable that will not fit into our hack machine. This is typically what you are going to face in today's what we call as Internet of things era, in Internet of things era we are going to see small embedded systems and these embedded systems are going to have very less memory, so what we call as memory footprint, the memory footprint of your program should be small that it fits into small embedded memory right you will have something like that like 2 kilobytes, yes 64 kilobyte, 256 kilobytes and you have to put the whole thing inside this and execute.

So one of the biggest effort in embedded programming that anyone of you... If you really go into a challenging problems in embedded programming in the IOTL I hope many of you will go and you take inspiration from this course I will be extremely happy, so if you basically do that then one of things that you will really land up there is that you need to start writing code that will fit into small memory foot prints right, so we will in the next 15 minutes of this course than is pending we will just go and see how you do such type of writing okay.

So even before looking into it in modern compilers like when you take a full-fledged compiler course you will learn the dual of the static linking what we call as dynamic linking right you will learn something call dynamic linking, what is dynamic linking? So when you compile a program like I have Main.vm, Bat.vm, Ball.vm I will just do a static linking of only Main, Bat and Ball because they are all part of my code then basically I will link them, I will assemble them and then I will convert it into hack code, so I will have the machine code for Main, Bat and Ball right that will be loaded into the memory, so I will have one executable which have only main bat and ball okay.

Now when this program is executing whenever it calls a OS routine like array.new or whatever the moment it starts calling that will go to the operating system and the operating system will fetch that array.new function instead of that being belonging to the original code, the operating system will fetch it into the memory that function alone executed and throw it back, return it back. So on demand I can link with that function get it executed after that I do not need it I can throw it back again whenever I need I can bring it so it is not that I can bring only that function for example if you take string.new already we saw string.new will call memory.alloc okay.

So memory.alloc me call sys.error so I bring if I want to execute string.new, first I will bring string.new I will start executing it in the middle when there is memory.alloc needed I will bring memory.alloc and execute that and after it finishes I will throw off memory.alloc then I will go...so memory.alloc in turn will ask (())(25:34) sys.error I will bring sys.error. The sys.error once this is getting over I will throw it off and then memory.alloc when it gets over, so on demand I will bring the functions from the storage to the memory and then return it back, so my memory footprint can essentially remain small, so this is called dynamic linking because we are linking to the functionalities of the operating system at execution.

So the dynamic word comes because we are doing it at the time of execution okay but in this course whatever we have done so far we do not have an operating system of that calibre which can help us do as dynamic linking. Typically all the IOT devices the small Internet of things devices that stays on your survey (())(26:31) camera at your house or it stays in the sense are that is basically measuring temperature in your air conditioning unit or wherever. Those things cannot afford to have an operating system, so we have to basically write small sea code like what we are doing, it will be typically like a hack machine, so how do we basically write code for hack such machines, so this is one case study that we will like to do now as a part of that remaining 2 modules. Thank you very much.