Operating Systems and Networks

Lecture 01:
Introducing the course
Behzad Bordbar

Introducing the course

- ❑Why we created this new module
  - Personal experience and trends in technology
  - HoS

Important

- ❑First time
- ❑Both practical and theory
- ❑Requires lots of work and time… exam HARD ☹

But

- ❑Does not require lots of background

Introducing the course (continue)

1. Demonstrate an understanding of the **fundamental** concepts and issues involved in OS and networking of IP-based systems Interact and manage an Operating System
2. Demonstrate an understanding of the challenges involved in the design of Distributed Systems in general and main methods of addressing them
3. Explain Transport Layer protocols and their differences
4. Understand the basics and practical issues and architectures involving in important Application Layer protocols
5. Demonstrate **practical understanding** of the theoretical foundations of Operating Systems and Distributed Systems

Introducing the course (continue)

- ❑Sessional: 1.5 hr examination (80%), continuous assessment (20%).
- ❑Supplementary (where allowed): 1.5 hr examination only (100%).
- ❑ You will have lab hours to ask your questions from a teaching assistant (time will be announced later) .

What am I going to learn?

Mixture of theory AND practice of:

- OS Fundamentals and architecture
- Shell programming
- OS networking
- OS support for Distributed Systems
- Distributed object, RMI and RPC
- Virtualisation, Xen, KVM *
- cryptographic algorithms and Security
- P2P (*)
- Wireless protocols
- Web servers: Apache server and nginx
- subversion and git
- maven (*)
- Distributed file systems ()
- HDFS
- Web Services (*)
- NoSQL and OS (*)

But we don't teach kernel programming

Exercise 0: learn Linux

- ❑**This course is not for you if you don't want to learn Linux**
- ❑Mac people are OK, as have access to shell.
- ❑Cygwin and MS powershell wont do ☹
- ❑Is Linux a new OS to you or need to brush up?
  - ❑SoCS machines
  - ❑Dual boots system
  - ❑Virtualised environment (install Ubuntu on **Vmware player** or **Virtual box**)

Exercise 0: Learn Basic Linux commands, for example from

http://www.debianhelp.co.uk/commands.htm

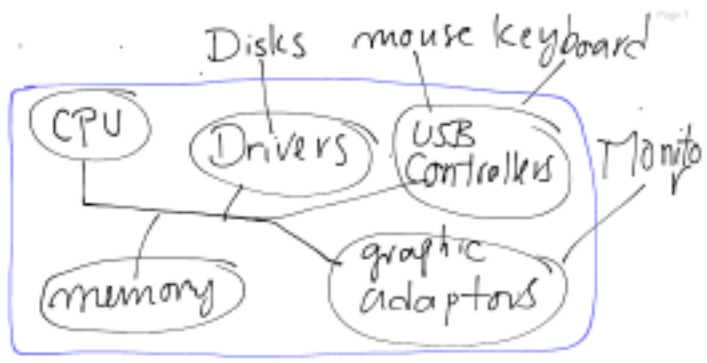
What reading material can help me?

- excellent library with lots of books on OS and Networking and Distributed systems
- Lots of online books

- Modern Operating Systems by Andrew S. Tanenbaum
- Operating System Concepts by Abraham Silberschatz, Peter B. Galvin and Greg Gagne
- Distributed Systems: Concepts and Design by George Coulouris, Jean Dollimore, Tim Kindberg and Gordon Blair
- Data Communications and Networking by Behrouz Forouzan

Operating system: preliminaries

- What is in your computer?



- We want to learn these? But what is OS?

central processing unit (CPU)

- "brain" of the computer

CPU

Semiconductor companies:

- Intel
- PC: core i7, i5,....
- Sever and Workstation processor: Intel Xeon* processor E7, E5, ...
- Mac: Hawell, Ivy Bridge, Sandy Bridge,...

AMD (Advanced Micro Devices Ltd)

- Boldddozer (Vishera, Zambezi,...), K10 series (Athlon, Operon, ...)

All above are based on x86: backward compatible (?) set architecture based on Intel 8086 cpu.

AMD also produces CPU based on a blueprint developed by ARM

- Majority of mobile phones use ARM

CPU(continue)

How does it work? *Draw picture!*

- fetch the first instruction from memory,
- decode it to determine its type and operands
- execute it, and
- then fetch, decode, and execute subsequent instructions.

Instructions are CPU specific

- i7* core can not execute ARM instructions!

CPU(continue)

Accessing memory to get an instruction or data takes much longer than executing an instruction:

- CPUs contain registers inside to hold variables and temporary results.

What is instruction set?

- Load/store data from/to memory into a register
- combine two operands from registers and store result, for example adding
- ...

Register

- 8-bit or 32 –bit storage. Examples of registers are:
- **General purpose** reg.: temporary data & results
- **program counter**: memory address of the next instruction to be fetched and then program counter is updated to point to its successor.
- **Stack pointer**: points to the top of the current stack in memory

Stack?

- Stack contains one frame for each procedure to be entered
- stack frame holds those input parameters, local variables, and temporary variables that are not kept in registers.

CPU modes

- At least two modes, **kernel mode** and **user mode**
- kernel mode: CPU can execute every instruction in its instruction set and use every feature of the hardware (complete access to hardware).
- When CPU in kernel mode we say OS in kernel mode.
- User programs run in user mode, which permits only a subset of the instructions to be executed and a subset of the features to be accessed. Generally, all instructions involving I/O and memory protection are disallowed in user mode.
- Program Status Word (PSW) is a register that (among other things) keeps the mode of the CPU

System call and Trap

So how do a user program do for example I/O?

- Ask OS: a user program must make a system call, which traps into the kernel and invokes the operating system.
- The TRAP instruction switches from user mode to kernel mode and starts the operating system.
- When the work has been completed, control is returned to the user program at the instruction following the system call.

We will look at this in details later! Jut one small point

Not every trap is caused by system calls!

- Some traps are caused by the hardware to warn of an exceptional situation such as an attempt to divide by 0 or an arithmetic underflow.
- Trap causes operating system gets control and must decide what to do. Example:
  - OS terminates program when error
  - error can be ignored and underflowed number set to 0
- Do you know about exception handling: that is when control is handled to program.

GPU =/=CPU

- Graphic Processing Unit (GPU) coined by Nvidia
- Called VPU (Visual Processing Unit) by ATI (a competitor of Nvidia)
- GPU: electoronic circuit specialised for graphic processing
- Presented as a video card in a PC for processing graphics
- Originally designed and used for image manipulation. Turned out to be suitable for parallel computing (most notably CUDA)

How to say what CPU you have

```
$ cat /proc/cpuinfo
processor : 0
vendor_id : GenuineIntel
cpu family : 6
model : 58
model name : Intel(R) Core(TM) i7-3667U CPU @ 2.00GHz
stepping ....
```

- You see a number i7-3667U, what does it mean?

See intel page

- or run hardware lister

```
$ sudo lshw
$ sudo lshw|grep -i cpu
```

What happens when your computer starts?

Bootstrap program (bootloader, GRUB) runs first :

- It initializes all aspects of the system, from **CPU registers** to **device controllers** to **memory contents.**
- Load operating system and kernel to memory and start it.

Where is it saved if no CPU powered up?

- firmware: hardware in **read-only memory (ROM)** or **electrically erasable programmable read-only memory (EEPROM)**,
- Why in firmware? Can not be infected easily with virus

After your computer started:

- program at boot time become System Daemons/processes (in unix "init")
- Event occur (mouse click, a program want to access a file...) we refer to these as **interrupts** from either the hardware or the software.
- Hardware may trigger an interrupt at any time by sending a signal to the CPU.
- Software may trigger an interrupt by executing a special operation called a system call... (we see what happens with this, how about hardware)

When CPU interrupted:

- it stops what it is doing and immediately transfers execution to a fixed location.
- The fixed location usually contains the starting address where the service routine for the interrupt is located.
- The interrupt service routine executes
- on completion, the CPU resumes the interrupted computation.

Memory

An array of bytes.

- read-only memory (ROM) and EEPROM
- ROM can not be modified: suitable for bootstrap program or game cartridges
- EEPROM can be modified but not frequently: smartphones have EEPROM to store their factory-installed programs.
- CPU needs read **and write**: **main memory** (also called random-access memory, or RAM).
- Main memory commonly is implemented in Dynamic Random-Access Memory (DRAM) technology.

Memory (continue)

- Register is another type of memory.

Main memory goes away when machined turned off:

- Secondary storage: magnetic disk, optical disk, tapes

Cache

- The data that has been used a lot is cached in a faster storage system.
- So if CPU looking for info, first check cache then main memoery. ...

[cache is a concept more general than operating system, your browser has a cache, your dbms has a cache]

Speed of access to memory

Level	1	2	3	4	5
Name	registers	cache	main memory	solid state disk	magnetic disk
Typical size	< 1 KB	< 16MB	< 64GB	< 1 TB	< 10 TB
Implementation technology	custom memory with multiple ports CMOS	on-chip or off-chip CMOS SRAM	CMOS SRAM	flash memory	magnetic disk
Access time (ns)	0.25 - 0.5	0.5 - 25	80 - 250	25,000 - 50,000	5,000,000
Bandwidth (MB/sec)	20,000 - 100,000	5,000 - 10,000	1,000 - 5,000	500	20 - 150
Managed by	compiler	hardware	operating system	operating system	operating system
Backed by	cache	main memory	disk	disk	disk or tape

Picture from Dinosaur book

Summary

- Description of the module

Started by preliminaries of OS

- CPU
- Register
- System call trap and interrup
- What happens when computer starts?
- Different types of Memory and their speed

Will continue with preliminaries of OS

Operating Systems and Networks

Lecture 03:

Introduction to OS-part 2

Behzad Bordbar

Recap

Lecture 1:

- CPU: how it works, user and kernel mode, use of register, cache, ...
- System call trap and interrup
- What happens when computer starts?
- Different types of Memory and their speed

Will continue with preliminaries of OS

Demo lecture

Contents

- Service view (provider of services) of the OS
- Shell
- Everything a directory
- mkdir, mv, cp,...
- Access control
- Find, grep
- |, >, >>, ; and their differences
- wget,...

Contents

- How does mouse and keyboard work?
- Device controller
- CPU multitasking
- Time sharing
- a short study of system calls
  - API

How does mouse, keyboard ...work?

- We said

Hardware may trigger an interrupt at any time by sending a signal to the CPU.

- But how?
- Devices interact via device controller connected through a common bus to CPU. Draw picture!
- small computer-systems interface (SCSI) controller.
- SCSI controller: hardware (card or chip) that allows SCSI storage device to communicate with the operating system using a SCSI bus

Device controller

- Maintains some local buffer storage and a set of special-purpose registers.
- Device controller moves the data between the peripheral devices that it controls and its local buffer storage.
- Operating systems have a device driver for each device controller.
- you download drivers! Manually or automtaically
- Device driver understands the device controller and provides the rest of the OS with a uniform interface to the device: copy the same no matter what device

Back to, How does I/O (mouse..) work?

- Device driver loads the appropriate registers within the device controller.
- Device controller examines the contents of the registers to determine what action to take (read char from k/b)
- controller starts the transfer of data from the device to its local buffer.
- when transfer of data is complete, the device controller informs the device driver via an interrupt that it has finished its operation.
- Device driver then returns control to the operating system, possibly returning the data or a pointer to the data if the operation was a read or status (success…)
- Exercise: draw a sequence diagram for yourself!

How does I/O (mouse..) work? (cont..)

- This form of interrupt-driven I/O is fine for moving **small amounts** of data
- Not suitable for bulk data movement such as disk I/O.

Instead: Direct Memory Access (DMA) is used that takes CPU out of the loop.

- After setting up buffers, pointers, and counters for the I/O device, the device controller transfers an entire block of data directly to or from its own buffer storage to memory, with no intervention by the CPU.
- Hence: only one interrupt is generated per block, to tell the device driver that the operation has completed, rather than the one interrupt per byte generated for low-speed devices.
- CPU is made free to do other things

Multitasking in CPU

- OS picks and begins to execute one of the jobs in memory.
- Eventually job may have to wait for some task, such as an I/O operation, to complete.
- OS switches to, and executes, another job.
- When that job needs to wait, the CPU switches to another job, and so on.
- Eventually, the first job finishes waiting and gets the CPU back.

Time-sharing:

- CPU executes multiple jobs by switching among them, but the switches occur so frequently that the users can interact with each program while it is running.

Time sharing

requires CPU scheduling of user tasks. But how?

- Each user has at least one separate program in memory.
- A program loaded into memory and executing is called a **process**. We will study this in details!
- When a process executes, it typically executes for only a short time before it either finishes or needs to perform I/O.
- I/O takes long long long time compare to execution! (look at the speed of access slides!)

Time sharing

- Time sharing: several jobs be kept simultaneously in memory.
- CPU scheduling: process of deciding which job is brought to memory to be executed, when there are not enough room.

Reasonable response time must be ensured:

1. processes are swapped in and out of main memory to the disk
2. use virtual memory: a technique that allows the execution of a process that is not completely in memory

- virtual-memory scheme enables users to run programs that are larger than actual physical memory. Further, it abstracts main memory into a large, uniform array of storage, separating logical memory as viewed by the user from physical memory. This arrangement frees programmers from concern over memory-storage limitations.

Dual mode

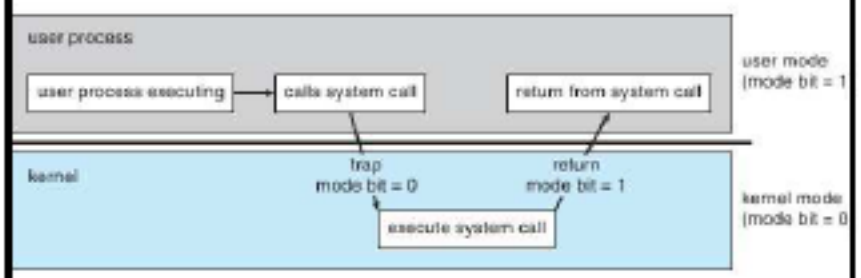


Figure from Dragon book.

- Dual mode OS protect from harm caused by privileged instructions
- Extended to multi mode by domains: Dom0, DomU

Why do we need hardware support?

- MS-DOS: Intel 8088 architecture, which has no mode bit
- user program can wipe out the whole OS
- programs are able to write to a device …

In dual mode:

- hardware detects errors that violate modes and handle them by Os
- stops user program attempts to execute an illegal instruction or to access memory of other users

When error detected

- OS must terminate the program
- OS gives error message
- produces memory dumps by writing to a file (users can check or OS vendors can check (Sun)).

system calls

provide an interface to the services made available by an operating system.

What language: are System-call written in?

- typically C and C++ and sometimes assembly-language involved
- explain the system calls for reading data from one file and writing to another file:

$cp file1 file2

open file1, possible error(print, abort), create file2 (file2 exists, rewrite/rename…), start read and write (errors:disk space, memory stick unplugged…), all read and written, close files, ack

Do I access system call directly?

system calls: API to wrap system calls

Application Programming Interface (API)

- specifies a set of functions that are available to an application programmer, including the parameters that are passed to each function and the return values the programmer can expect.
- programmer accesses an API via a library of code provided by the operating system.

Example of APIs:

1. Windows API for Windows systems

Example: CreateProcess() which invokes the NTCreateProcess() system call in the Windows kernel return value 0 or 1 (error)

system calls: API (continue)

Example of APIs:

2. POSIX API for POSIX-based systems (UNIX, Linux, and Mac OS X)

- programmer accesses an API via a library of code provided by the operating system.

Example: read

input:

- int fd: file descriptor to be read
- void *buf: pointer into buffer to be read into
- size_t count: maximum number of bytes to read

output:

- number of bytes read (if success)
- -1 if fail
- UNIX and Linux for programs written in the C language, the library is called libc.

system calls: API (continue)

Example of APIs:

3. Java API for programs that run on the Java virtual machine.

getParentFile()

invoked on a file object.

output:

Returns the abstract pathname of this abstract pathname's parent, or null if this pathname does not name a parent directory.

JVM uses the OS system calls.

why do we use API?

Why not invoking actual system calls directly?

- Program portability: program can compile and run on any system that supports the API
- system calls can often be more detailed and difficult to work with
- give access to high level objects (java API)

Do you know interfaces in java? using system calls is like implementing an (or many) interfaces

What happens when a user prog. makes a system call

- caller only needs to know the signature!
- method call and parameters are passed into a registers
- values saved in memory for example on table or stacks but addresses in registers
- Stack is preferred because do not put limit on the number of parameters stored.

summary

- Study of I/O devices
- Device Controller
- How OS manages multiple tasks
- System call and their use in user programs
- API and examples of API
- Similarity between API and interface

Operating Systems and Networks

Lecture 04:
Introduction to OS-part 3
Behzad Bordbar

Recap

- Study of I/O devices
- Device Controller
- How OS manages multiple tasks
- System call and their use in user programs
- API and examples of API

Contents

- Different types of system call
- process
- process =/= program
- Stack and heap
- process control block

Different types of system call

- process control
- file manipulation
- device manipulation
- information maintenance
- communications
- protection.

System call: process control

process =/= program
an instance of a running program (two process can be associated to the same program).

- end, abort
- load, execute
- create process, terminate process
- get process attributes, set process attributes
- wait for time
- wait event, signal event
- allocate and free memory

We will discuss some of these.

end and abort

- Stopping a running program:
- normal: end()
- abnormally: abort().
- abnormally: division to zero, file not available:
- a dump of memory is created for
- debugging (code has problem)
- finding cause (unauthorised access)

Exercise: read about stdout and stderr

load(), execute(), createProcess()

- $firefox will result in loading of firefox from another program (terminal)
- Two issues
- what happens when loading program terminates?
- should I keep the current program running? (backgroud program &)

Creating and managing processes

- If we create a process we can control its execution
- getting attributes
- setting attributes
- maximum allowable execution time
- terminate process if we find that it is incorrect or is no longer needed
- wait for an event to happen wait_event() and take action signal_event()

Critical section

- this file is being modified by another program!
- Critical section: two or more proccess access the same data… one must access at a time
- operating systems often provide system calls allowing a process to lock shared data. Then, no other process can access the data until the lock is released.
- acquire_lock() and release_lock()

Hard topic: semaphore and monitors

File Management

- create file, delete file
- open, close
- read, write, reposition
- get file attributes, set file attributes

Device management

process may need multiple resources to execute:

Example: access to particular disk

- request device, release device
- read, write, reposition
- get device attributes, set device attributes
- logically attach or detach devices

Information maintenance

- get time or date, set time or date
- get system data, set system data
- get process, file, or device attributes
- set process, file, or device attributes

Communications

- How does two process communicate?
- message-passing model
- shared-memory model

message-passing model

- Java program (one process) connecting a database (second process) to read/write data (JDBC)
- Before message, a connection must be opened.
- Naming: host id and id of the process running on the host
- Connection created and open()
- permission for communication to take place with an accept connection() call.
- processes involved are called client and server
- read_message() and write_message() system calls.
- close_connection() sys. call terminates communication.

shared memory

- shared black board model
- processes gain access and read and write to a shared part of memeory
- operating system prevent one process from accessing another process's memory. For shared memory two or more processes agree to remove this restriction.
- Who ensures two process do not read and write same location simultaneously: the two processes.
- Thread (thread =/= process =/= program)

Communications

No matter shared-memory or message-passing, the following system calls are supported:

- create, delete communication connection
- send, receive messages
- transfer status information
- attach or detach remote devices

protection

- mechanism for controlling access to the resources: file, devices, processes, ...
- Mostly designed around access control (remember practical lecture)
- get permission
- set permission
- allow_user
- deny_user

system program vs applications

System programs (utilities)

- convenient environment for program development and execution
- interfaces to system calls

Examples:

- File management.
- Status information
- File modification.
- Programming-language support...

system program vs applications

OS are often supplied with applications : programs that are useful in solving common problems or performing common operations.

Example: database, compiler, game...

process

- A program is a passive entity.
- a file containing a list of instructions stored on disk(foo.exe).
- a process is an active entity, with a program counter specifying the next instruction to execute and a set of associated resources.
- A program becomes a process when an executable file is loaded into memory.
- what processes I am running?

$ps –el

What are these resources?

process in memory

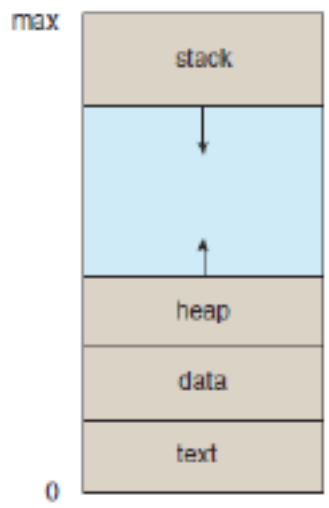
A process has the following:

- text section: the program code, which is sometimes known as the text section.
- current activity: the value of the program counter + the contents of the processor's registers.
- process stack: contains temporary data (such as function parameters, return addresses, and local variables)
- data section: contains global variables.
- heap: which is memory that is dynamically allocated during process run time.

heap and stack

- stack frames

picture from dragonbook



stack and heap

- Stack is a contiguous block of memory that is allocated to each process.
- Stack is LIFO (last in first out).
- Stack has stack frame: contains parameter to functions, local vairables and data for recovering previous stack frame.
- When a function is called frame for that function is pushed into stack, when done executing we pop. When we call stack grow, when we execute stack shrinks and we end up with the caller and then frame pops.

stack and heap

- Stack is high performance and is fixed-rate
- in C/C++ and java depending on your code goes to

Java

- new book() puts book on heap frame
- int price put price on stack frame

C/C++

- int familymemeber[10] stack
- int* myArray= new int[10]; puts on heap

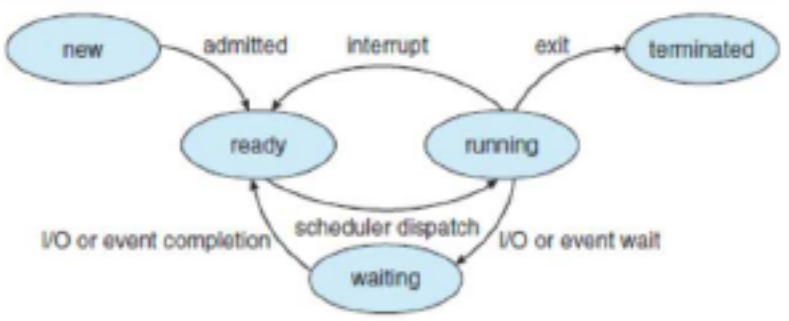
stack and heap

- fast register that we read from memory into them and do the computation and push back to memory. They are very fast.
- Stack register
- ESP (stack pointer) points into top of stack manipulated by pop and push and call (FUNCTION) and RET
- EBP (stack base pointer): this one points to star tof the frame. ESP shows offset from EBP
- EBP fixed and ESP is dynamic
- ESI and EDI source and destination instructions. Where to go? Who called who?

running a process and java

- How do I load a process to memory?
- java is example of a process that is an execution environment for other code.
- JWM is a process that interprets the loaded Java code
- loaded java byte code?
- $javap –c progname

process state



- picture from dragonbook

process control block (PCB)

information required for context switching

- Process state
- Program counter.
- CPU registers
- CPU-scheduling information
- Memory-management information
- Accounting information
- I/O status information

summary

- various types of system call
- Process and differences between process and program
- stack and heap
- process state and control block

... now we know what is saved when switching context.

Operating Systems and Networks

Lecture 05:
Introduction to OS-part 4
Behzad Bordbar

Recap

- Different types of system call
  - process control
  - file manipulation
  - device manipulation
  - information maintenance
  - communications
  - protection
- process
- process =/= program (and =/= thread)

Contents

- Heap vs stack (what size is a proc stack?)
- proc. states and control block
- context switching
- process and thread
- process in linux

From C to Assembly

- http://gcc.godbolt.org/
- What does this piece of C code do?

```
#include <stdio.h>
 #define LAST 10
  int main()
  {
     int i, sum = 0;
     for ( i = 1; i <= LAST; i++ ) {
       sum += i;
     }
     printf("sum = %d\n", sum);
     return 0;
  }
```

From C to Assembly (continue)

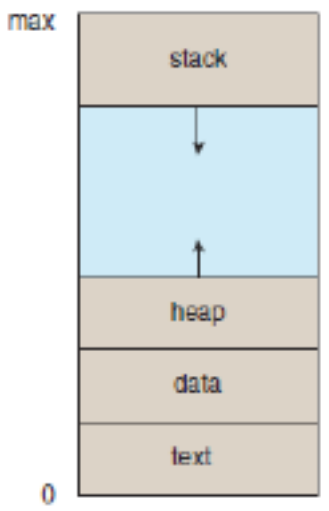
- Explore assembly output with different compilers (remove – O2 from Compiler options and activate Colourise)
- process a program which is running.
- What do we need for that?
- observe: push, pop, mov…

stack and heap

- Stack is a contiguous block of memory that is allocated to each process.
- Stack is LIFO (last in first out).
- Stack has stack frame: contains parameter to functions, local variables and data for recovering previous stack frame.
- When a function is called frame for that function is pushed into stack, when done executing we pop. When we call stack grow, when we execute stack shrinks and we end up with the caller and then frame pops.

heap and stack

- stack frames



picture from dragonbook

What is difference between heap and stack?

stack

- in java we have a data type jave.util.stack (not here!)
- part of computer memory that stores temporary variables created by each function (including the main() function). T
- FILO data structure managed and optimized by the CPU
- push and pop of stack frames
- Once a stack variable is freed, that region of memory becomes available for other stack variables.

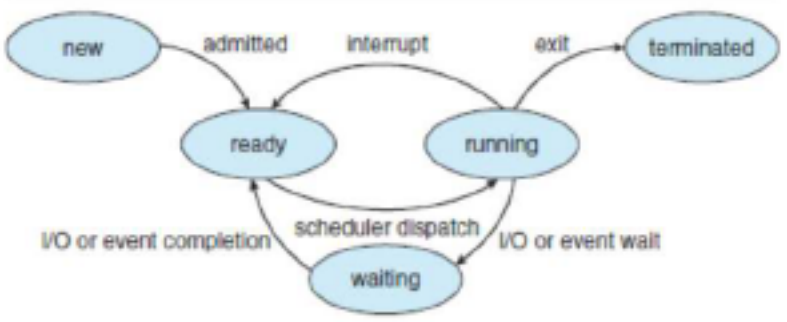
stack

- advantage: that memory is managed for you [no need to allocate memory yourself or free it once you don't need it any more]
- CPU organizes stack memory so efficiently (very fast).
- stack is limited
- $pgrep chrome
- $ cat /proc/<PID>/limits

heap

- part of computer memory that is not managed automatically.
- in C you must use malloc() or calloc() to allocate
- you are responsible for using free() to de-allocate that memory
- if you don't: memory leak (memory on the heap will still be set aside (and won't be available to other processes)
- Unlike the stack, the heap does not have size restrictions on variable size (apart from the physical limitations of your machine).
- Heap memory is slightly slower to be read from and written to, because one has to use pointers to access memory on the heap.
- Heap variables are essentially global in scope

process state



picture from dragonbook

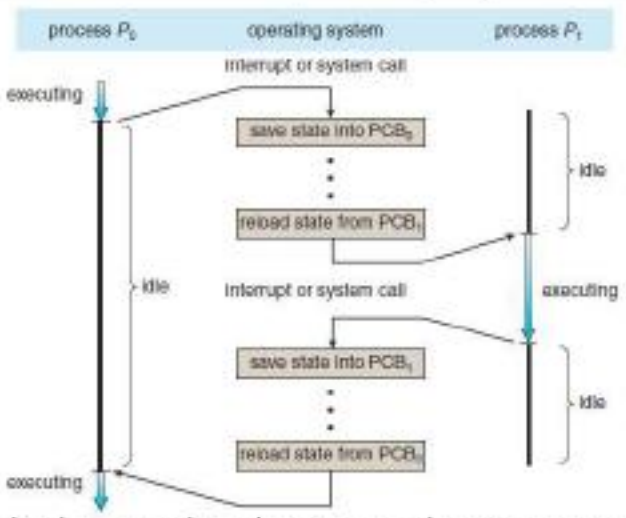
- only one process can be running on any processor at any instant.
- M any processes may be ready and waiting,

process control block (PCB)

information required for context switching

- Process state
- Program counter.
- CPU registers
- CPU-scheduling information
- Memory-management information
- Accounting information
- I/O status information

How does CPU execute multiple processes?



dragonbook. so… how do process and program compare?

Multiprogramming (revisited)

- Maximising CPU utilisation: some process running at all time
- time sharing: is to switch the CPU among processes so frequently that users can interact with each program while it is running
- Only one process can run at a time, the rest of processes must wait

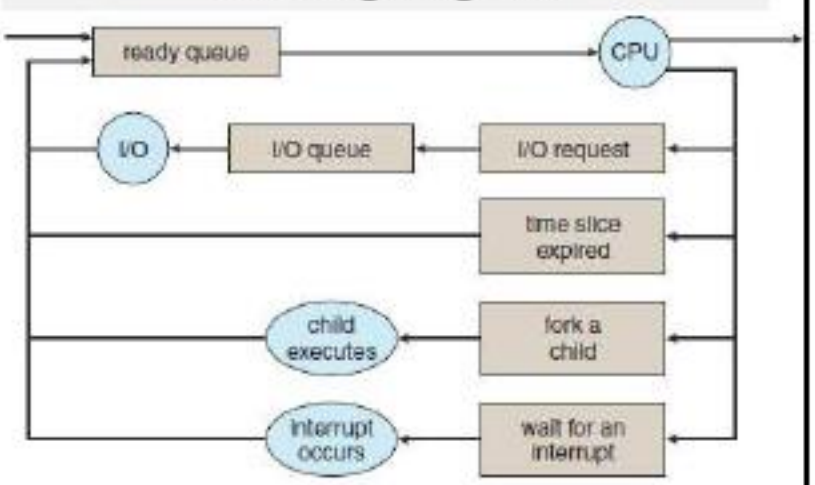
Process Scheduler:

- selects an available process for program execution on the CPU.

Queues

- job queue: all processes in the system
- ready queue: processes that are residing in main memory and are ready and waiting to execute are kept on a list (linked list)
- PCBs that have a link to the next PCB
- Device queue: list of processes waiting for a particular I/O device is called a

Queuing Diagram



source Dragon book

Context switching

- interrupt >> a state save of the current state of the CPU and then a state restore to resume operations
- kernel saves the context of the old process in its PCB and loads the saved context of the new process scheduled to run.
- Context-switch time is pure overhead.
- Switching speed varies from machine to machine (few milliseconds)

Process and Thread

- McDonald on high street: thread and process
- Example: a document with 1) editor 2) spell checker
- fork in unix (two process)
- can we share the context… thread
- Suitable particularly for multicore

Process in Linux: called task

- processes form a tree $ps -el
- unique id number (pid integer)
- The init process (pid of 1) is the root parent and spwans other proceses
- kthreadd: creating additional processes that perform tasks on behalf of the kernel

- ps -ef |more
- ps -efH |more
- pstree

Process in Linux

- fork() system call: makes new process
- New process has a copy of the address of the original process. Why?
- better communication between father & child
- father continues executing and child calls exec() sys. call and replaces memory space with new prog.
- Draw picture + example of shell and editor
- both proc. can create more children or
- father invokes wait() sys. call to move to ready queue until the termination of the child (background proc. &)
- termination, cascading termination, zombie process

Process in Linux

- termination: A process terminates when it finishes executing its final statement and asks the operating system to delete it by using the exit() system call.

Then what happens:

- return status value (typically an integer) to its parent process (via the wait() system call)
- All the resources of the process—including physical and virtual memory, open files, and I/O buffers—are deallocated by the operating system.

A process can be terminated directly by other process

- cascading termination: if parents terminate all children terminated (show via shell and background process)

Process in Linux (continue)

- What is exit status?
- wait() system call by parents
- a pointer is passed to return exit status of child
- zombie process
- orphan process
- terminating of orphan process by init()

summary

- Heap vs stack (what size is a proc stack?)
- proc states and control block
- context switching
- process and thread
- process in linux

Operating Systems and Networks

Lecture 07:

Introduction to OS-part 5

Behzad Bordbar

Recap

- General recap of all we have learnt!
- CPU, how computer starts? kernel/user mod, system calls, mutlitasking

Last week

- Heap vs stack (what size is a proc stack?)
- proc states and control block
- context switching (just overhead ☹)
- process and thread
- process in linux

Contents

- why do we need threads?
- What is a thread?
- What is multicore(multiprocess)?
- How does it fit into the story.
- Is more core ALWAYS better?
- How are threads implemented?
- End… move to networking

What is a thread?

- program=/=process =/= thread
- consider client's accessing a server. Design a model for interaction?

Modern OS are multi-threaded, multiple threads operate in the kernel, and each thread performs a specific task:

- managing devices
- managing memory
- interrupt handling.

What is a thread? (continue)

- program=/=process =/= thread
- Have you written a multi-threaded program?

main() + gc …

- Garbage collection!
- if GUI many more

thread is a basic unit of CPU utilization

Single threaded process vs. multi-threaded process.

- Why not multiple processes?

Data can be shared, but execution separated!

What is a thread? (continue)

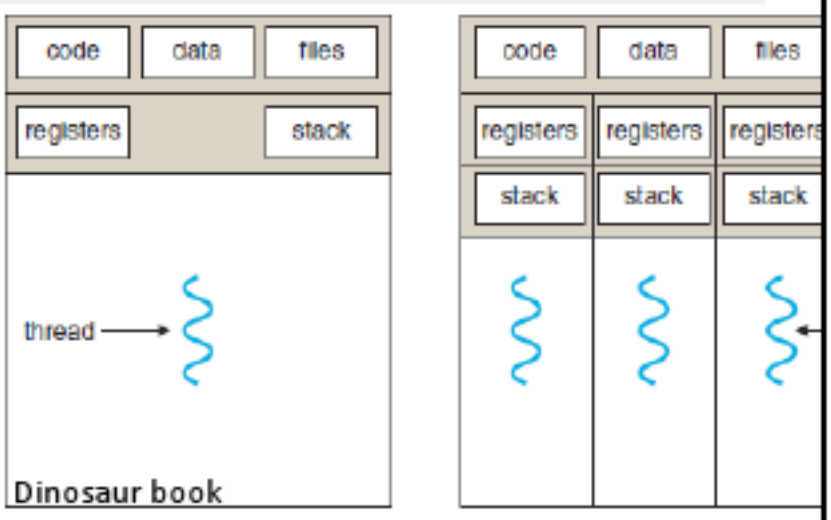
Similar to process, a thread has

- thread ID,
- program counter
- register set
- stack

threads belonging to same process share

- code section
- data section
- OS resources, such as open files

single threaded vs multithreaded



How to see threads in Linux

- ps -e -T | grep firefox
- -e all processes
- -T all threads
- Exercise: find out threads for a number of well know processes
- *How many threads are running on your machine?*

Why threads?

- Responsiveness

a time consuming operation or lengthy process not blocking the whole process

Single threaded GUI may block the usage

- Resource sharing

Processes: shared memory and message passing (program writes code for them)

threads share the memory and the resources of the process to which they belong by default.

Why threads? (continue)

- Economy.

Allocating memory and resources for process and context switching is computationally costly

threads share the resources of the process; more economical to create and context-switch threads.

[in some cases creating a process is about thirty times costlier and switching context is five times slower]

- Scalability

multicore allows shared processing, so multi-threading is much faster than muli-processing

multi process, multicore and all that

- multiprocess: many CPU chip

Symmetric and asymmetric design

- multicore: single CPU chip has multiple computing core (register, cache…)
  - faster communication (as no inter process communication)
  - significantly less energy consumed

Be aware: people use two phrases interchangeably!

- blade server (data centre and Cloud):

processor process, I/O boards, and networking cards are placed in the same chassis

multicore

- single core one thread executing at at time, two cores two threads…
- Single core illusion of parallelism (by fast switching) , multiple core true parallelism
- in single core task run concurrently not parallel

Amdahl's law

- if I add core will I always make execution faster?

speed up $\leq \frac{1}{S+\frac{1-s}{N}}$

- s percentage of portion that are serial
- N number of processing cores

what happens if N becomes large ($N \rightarrow \infty$)....

throwing in more core is not going to solve the problem always!

You may need to change the program!

110

Threads an operating system

- user level threads
- kernel level threads (managed by kernel support)

What is the relationship between the two groups:

- many-to-one model
- one-to-one model
- many-to-manymodel.

111

many to one

- multiple user-level threads to one kernel thread
- Thread management is done in user space so it is efficient

Not used widely any more:

Only one thread can access kernel at a time

- All involving process block if a thread makes a blocking system call
- multiple threads are unable to run in parallel on multicore systems.

112

one-to-one

- linux and window use this
- Each user thread is mapped to a kernel thread
- When a thread makes blocking system call, another thread can run.
- multiple threads can run in multiprocessors.
- But resource hungry and burden on performance
- Upper bound on the number of threads
- What is the maximum number of threads allowed on my machine?

cat /proc/sys/kernel/threads-max

113

many-to-many model

- many user-level threads are handled by multiple kernel threads.
- Developer can create as many user thread
- kernel can schedule one thread create maximum concurrent user threads is bounded by number of kernel threads
- when threads run in parallel on a multiprocessor there is advantage
- when one thread performs a blocking system call, the kernel can schedule another thread for execution.

114

How to program threads?

thread library: API for creating and managing threads.

1. A library entirely in user space with no kernel support i.e. a local function call in user space and not a system call.
2. a kernel-level library supported directly by the operating system, i.e. code and data structures for in kernel space.

- Invoking a function in the API for the library typically results in a system call to the kernel.

115

How to program threads? (continue)

main thread libraries:

1. Windows (uses kernel level library on Windows)
2. POSIX Pthreads (both user and kernel level)

posix? [
(Portable Operating System Interface)
family of standards by IEEE, ensure compatibility
Unix like, but microsoft supports some parts, why?]

cygwin posix compliant

3. Java threads: implemented using a thread library available on the host system[windows or pthread]

- Java threads are object: implement runnable or extend thread…

communication and networking

- End of preliminaries of OS

Be aware:

- lots left to learn
- similarities/differences between OS
- some topics important and we did not study: (memory access, registry/hive,…)
- Communicating processes (on a machine and across)
- You know one method for processes to communicate???

pipe |

- command1 |command2 (both in window and linux | dir |more)
- pipes allow to process to communicate
- but how?
- a temporary file is generated on disk
- command1 writes into it and command2 reads?
- but how?
- standard input, standard output and standard error. (next lecture)
- ordinary pipe (anonymous pipe in Windows)
- named pipe (mkfifo) we dont study this.
-

Summary

- Motivated and studied the reason for threads
- threads are units of computation
- multicore is better for threads
- sequential part of core dictates how many core can be useful… need to change code to benefit from manycore
- everything in linux is treated as files even pipe
- further mystery!