CS 35L Software Construction Lab
Week 5 – System Calls

Kernel

- Core of OS software **executing in supervisor** state
- **Trusted software:**
  - Manages hardware resources (CPU, Memory and I/O)
  - Implements protection mechanisms that could not be changed through actions of untrusted software in user space

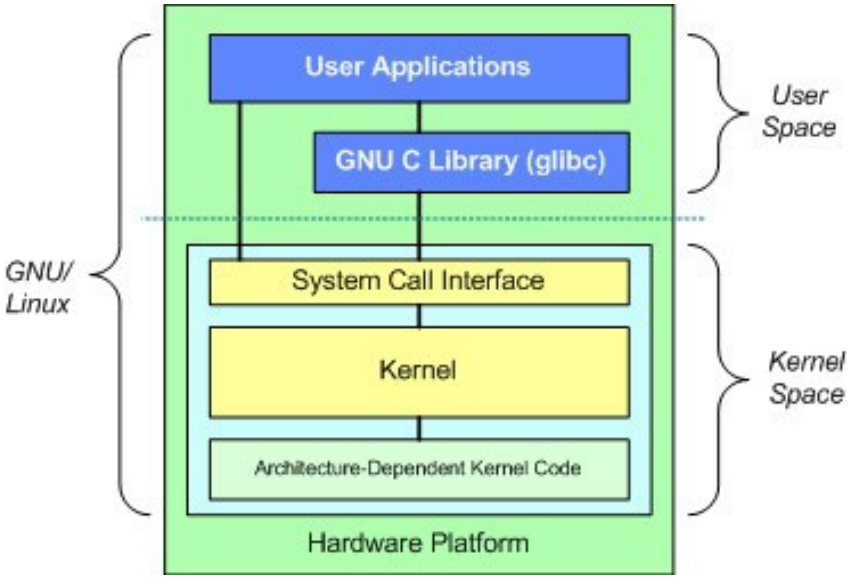


Image by: Tim Jones (IBM)

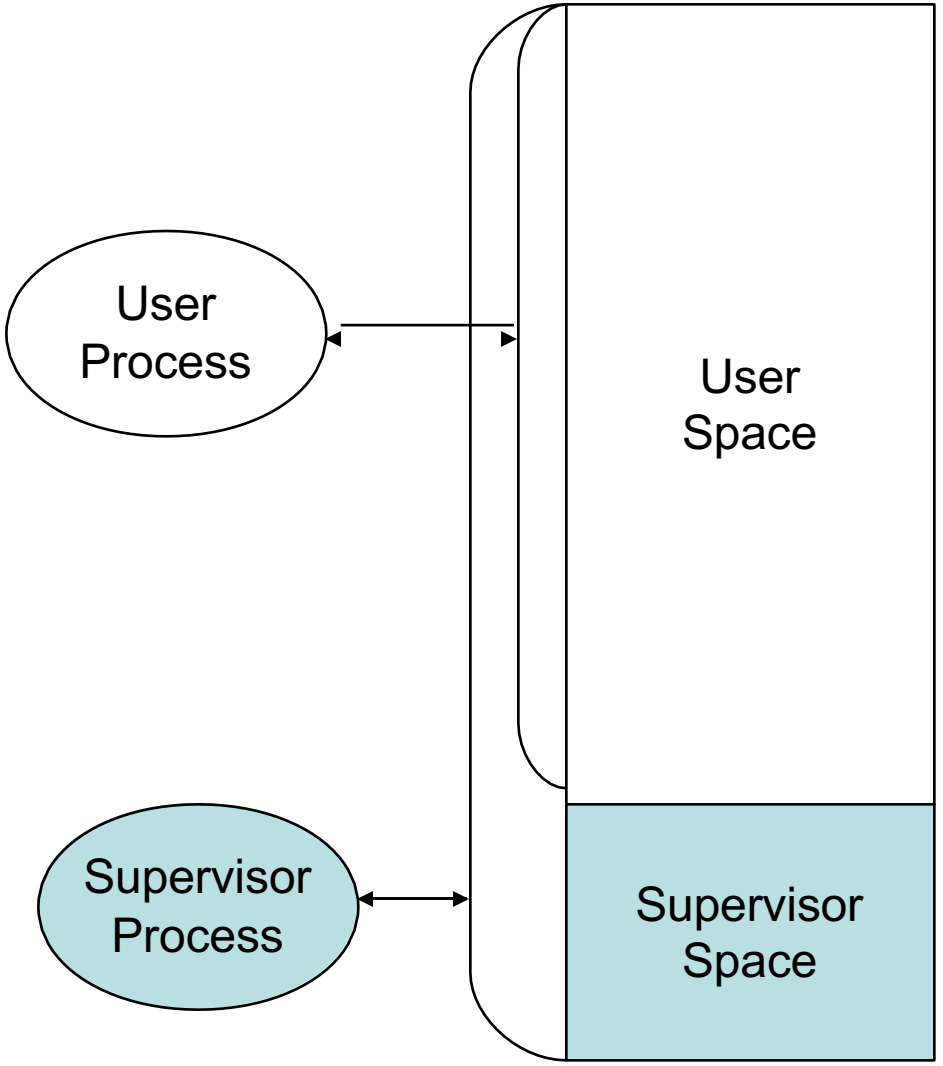
Goals for Protection and Fairness

- Goals:
  - **I/O Protection**
    - Prevent processes from performing illegal I/O operations
  - **Memory Protection**
    - Prevent processes from accessing illegal memory and modifying kernel code and data structures
  - **CPU Protection**
    - Prevent a process from using the CPU for too long

=> instructions that might affect goals are privileged and can only be executed by *trusted code*

Processor Modes

- Operating modes that place restrictions on the type of operations that can be performed by running processes
  - User mode: restricted access to system resources
  - Kernel/Supervisor mode: unrestricted access



User Mode vs. Kernel Mode

- Hardware contains a mode-bit, e.g. 0 means kernel mode, 1 means user mode

- User mode
  - CPU **restricted** to unprivileged instructions and a specified area of memory

- Supervisor/kernel mode
  - CPU is **unrestricted**, can use all instructions, access all areas of memory and take over the CPU anytime

Why Dual-Mode Operation?

- System resources are shared among processes

- OS must ensure:
  - **Protection**
    - an incorrect/malicious program cannot cause damage to other processes or the system as a whole
  - **Fairness**
    - Make sure processes have a fair use of devices and the CPU

Goals for Protection and Fairness

- Goals:
  - **I/O Protection**
    - Prevent processes from performing illegal I/O operations
  - **Memory Protection**
    - Prevent processes from accessing illegal memory and modifying kernel code and data structures
  - **CPU Protection**
    - Prevent a process from using the CPU for too long

=> instructions that might affect goals are privileged and can only be executed by *trusted code*

Which Code is Trusted?
=> The Kernel *ONLY*

- Core of OS software **executing in supervisor** state
- **Trusted software:**
  - Manages hardware resources (CPU, Memory and I/O)
  - Implements protection mechanisms that could not be changed through actions of untrusted software in user space
- **System call interface** is a **safe way** to expose privileged functionality and services of the processor

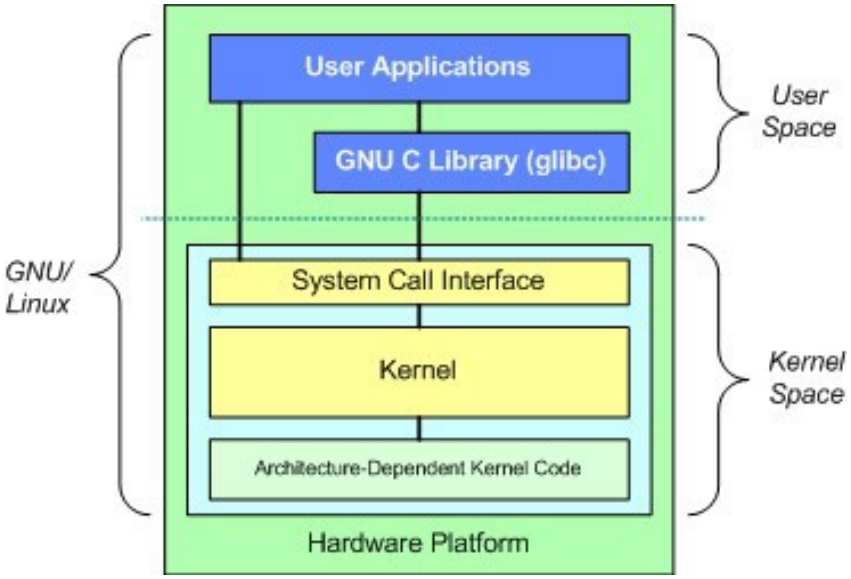
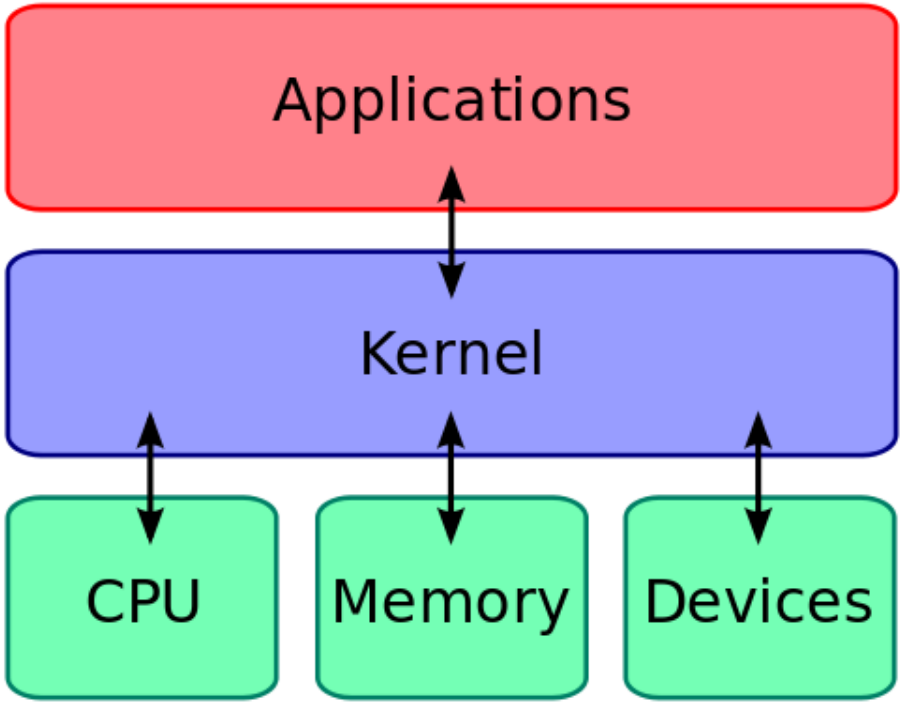


Image by: Tim Jones (IBM)

What About User Processes?

- The kernel executes privileged operations on behalf of untrusted user processes

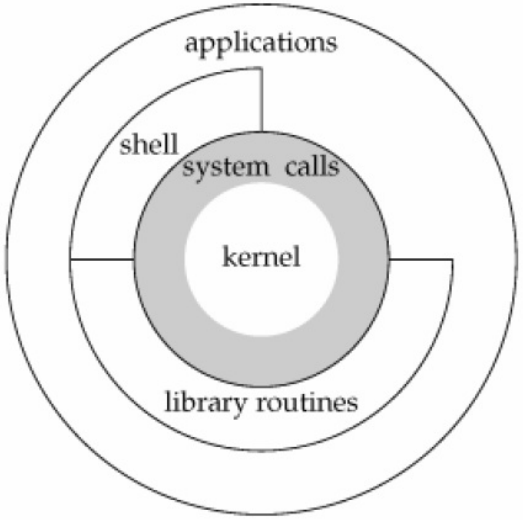


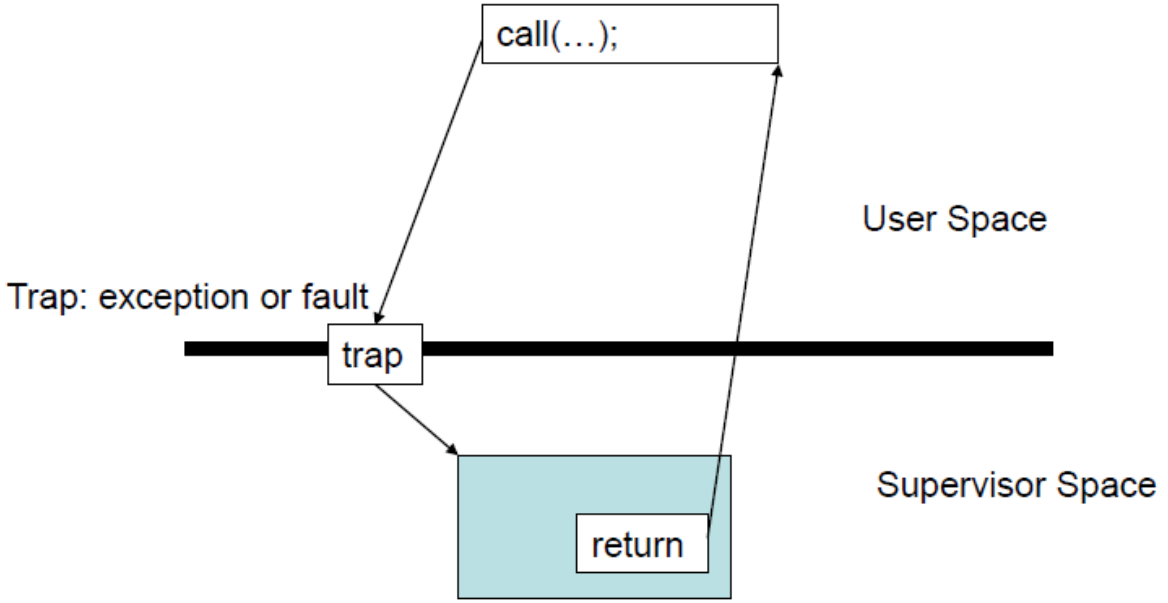
System Calls

- Special type of function that:
  - Used by user-level processes to request a service from the kernel
  - Changes the CPU's mode from user mode to kernel mode to enable more capabilities
  - Is part of the kernel of the OS
  - Verifies that the user should be allowed to do the requested action and then does the action (kernel performs the operation on behalf of the user)
  - Is the ***only way*** a user program can perform privileged operations

System Calls

- When a system call is made, the program being executed is interrupted and control is passed to the kernel
- If operation is valid the kernel performs it





System Call Overhead

- System calls are expensive and can hurt performance
- The system must do many things
  - Process is interrupted & computer saves its state
  - OS takes control of CPU & verifies validity of op.
  - **OS performs requested action**
  - OS restores saved context, switches to user mode
  - OS gives control of the CPU back to user process

Example System Calls

```
#include<unistd.h>
```

- `ssize_t` **`read`**`(int fildes, void *buf, size_t nbyte)`
  - fildes: file descriptor
  - buf: buffer to write to
  - nbyte: number of bytes to read
- `ssize_t` **`write`**`(int` *`fildes`*`, const void *`*`buf`*`, size_t` *`nbyte`*`);`
  - fildes: file descriptor
  - buf: buffer to write from
  - nbyte: number of bytes to write
- `int` **`open`**`(const char *pathname, int flags, mode_t mode);`
- `int` **`close`**`(int fd);`
- File descriptors
  - 0 stdin
  - 1 stdout
  - 2 stderr

Example System Calls

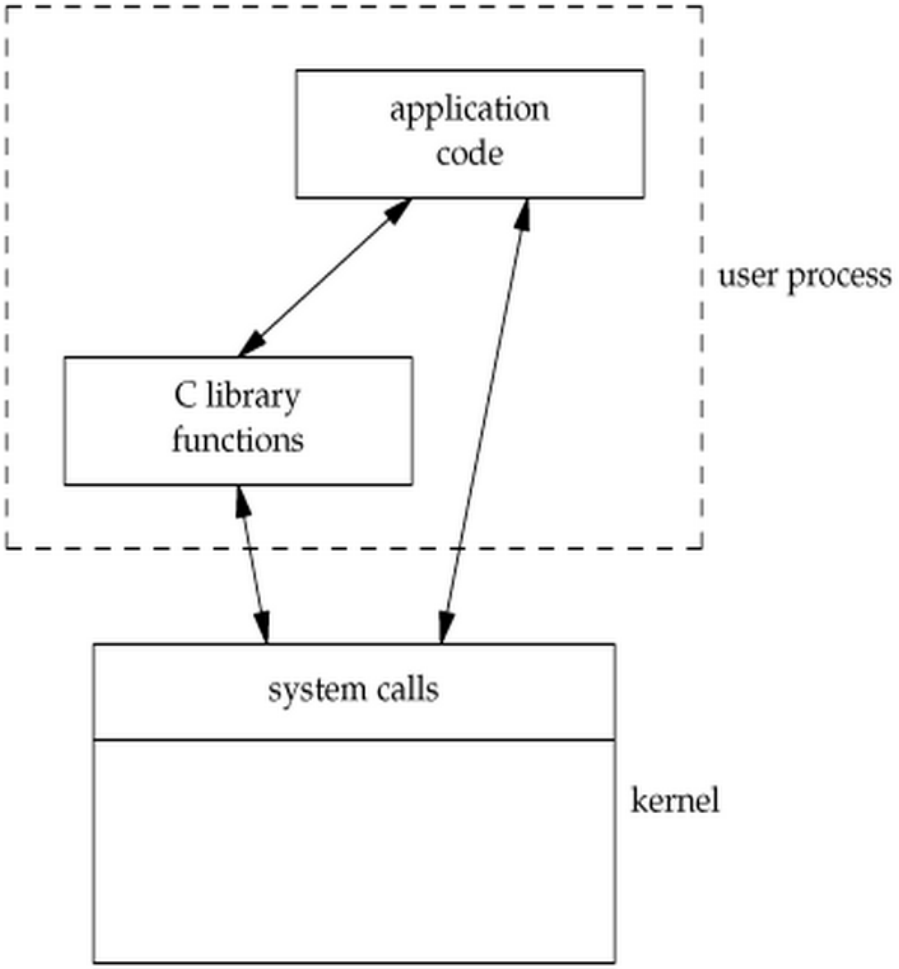
- `pid_t` **`getpid`**`(void)`
  - Returns the process ID of the calling process
- `int` **`dup`**`(int fd)`
  - Duplicates a file descriptor fd. Returns a second file descriptor that points to the same file table entry as fd does.
- `int` **`fstat`**`(int filedes, struct stat *buf)`
  - Returns information about the file with the descriptor filedes into buf

```
struct stat {
    dev_t     st_dev;     /* ID of device containing file */
    ino_t     st_ino;     /* inode number */
    mode_t    st_mode;    /* protection */
    nlink_t   st_nlink;   /* number of hard links */
    uid_t     st_uid;     /* user ID of owner */
    gid_t     st_gid;     /* group ID of owner */
    dev_t     st_rdev;    /* device ID (if special file) */
    off_t     st_size;    /* total size, in bytes */
    blksize_t st_blksize; /* blocksize for file system I/O */
    blkcnt_t  st_blocks;  /* number of 512B blocks allocated */
    time_t    st_atime;   /* time of last access */
    time_t    st_mtime;   /* time of last modification */
    time_t    st_ctime;   /* time of last status change */
};
```

Library Functions

- Functions that are a part of standard C library
- To avoid system call overhead use equivalent library functions
  - getchar, putchar vs. read, write (for standard I/O)
  - fopen, fclose vs. open, close (for file I/O), etc.
- How do these functions perform privileged operations?
  - They make system calls

So What's the Point?



- Many library functions invoke system calls indirectly
- So why use library calls?
- Usually equivalent library functions make fewer system calls
- non-frequent switches from user mode to kernel mode => less overhead

Unbuffered vs. Buffered I/O

- **Unbuffered**
  - Every byte is read/written by the kernel through a system call
- **Buffered**
  - collect as many bytes as possible (in a buffer) and read more than a single byte (into buffer) at a time and use one system call for a block of bytes

=> Buffered I/O decreases the number of read/write system calls and the corresponding overhead

Laboratory

- Write `tr2b` and `tr2u` programs in 'C' that transliterates bytes. They take two arguments 'from' and 'to'. The programs will transliterate every byte in 'from' to corresponding byte in 'to'
  - `./tr2b 'abcd' 'wxyz' < bigfile.txt`
    - Replace 'a' with 'w', 'b' with 'x', etc
  - `./tr2b 'mno' 'pqr' < bigfile.txt`
- `tr2b` uses **getchar** and **putchar** to read from STDIN and write to STDOUT.
- `tr2u` uses **read** and **write** to read and write each byte, instead of using getchar and putchar. The nbyte argument should be 1 so it reads/writes a single byte at a time.
- Test it on a big file with 5000000 bytes

```
$ head --bytes=# /dev/urandom > output.txt
```

`time` and `strace`

- **time [***options***] *command* [***arguments...***]**
- Output:
  - real 0m4.866s: elapsed time as read from a wall clock
  - user 0m0.001s: the CPU time used by your process
  - sys 0m0.021s: the CPU time used by the system on behalf of your process
- **strace**: intercepts and prints out system calls to stderr or to an output file
  - $ strace –o strace_output ./tr2b ‘AB’ ‘XY’ < input.txt
  - $ strace –o strace_output2 ./tr2u ‘AB’ ‘XY’ < input.txt

System call programming and debugging

Week 5

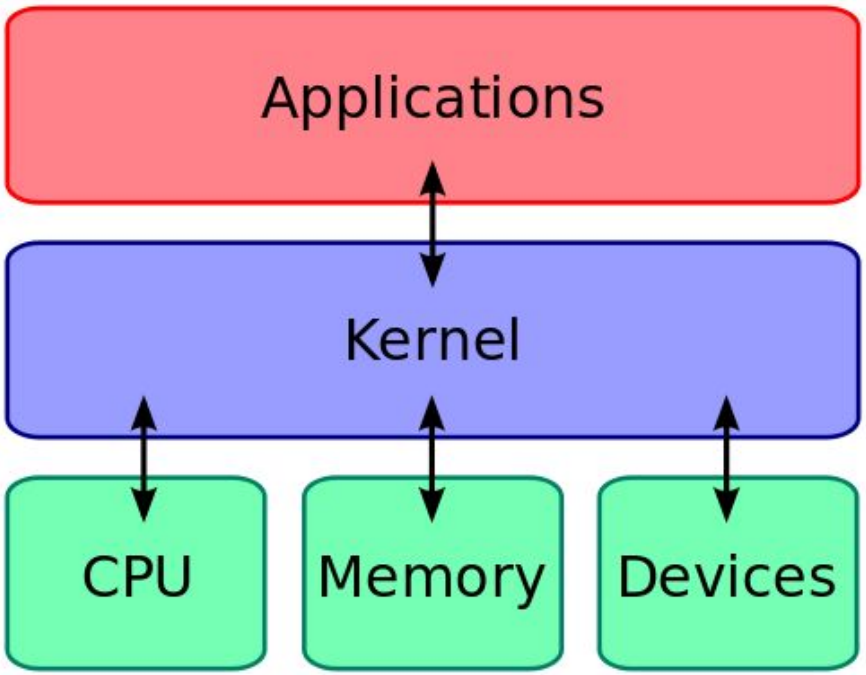
Presentation

- Slides
  - About 5 pages
  - Good motivation expected
  - Give audience one or two takeaways
  - Technical, structured and interesting
  - Due the night before you present

    (email to me and submit on CCLE)
- report
  - 800 ~ 1200 words
  - Structured like a technical paper
  - ACM format is a plus
- Students presentation material will be in final
  - Slides and report will not be shared

The Kernel

- Code of the OS **executing** in **supervisor** state
- Multiple applications running at the same time using time-sharing technique in cpu
- Achieve isolation and fairness among applications

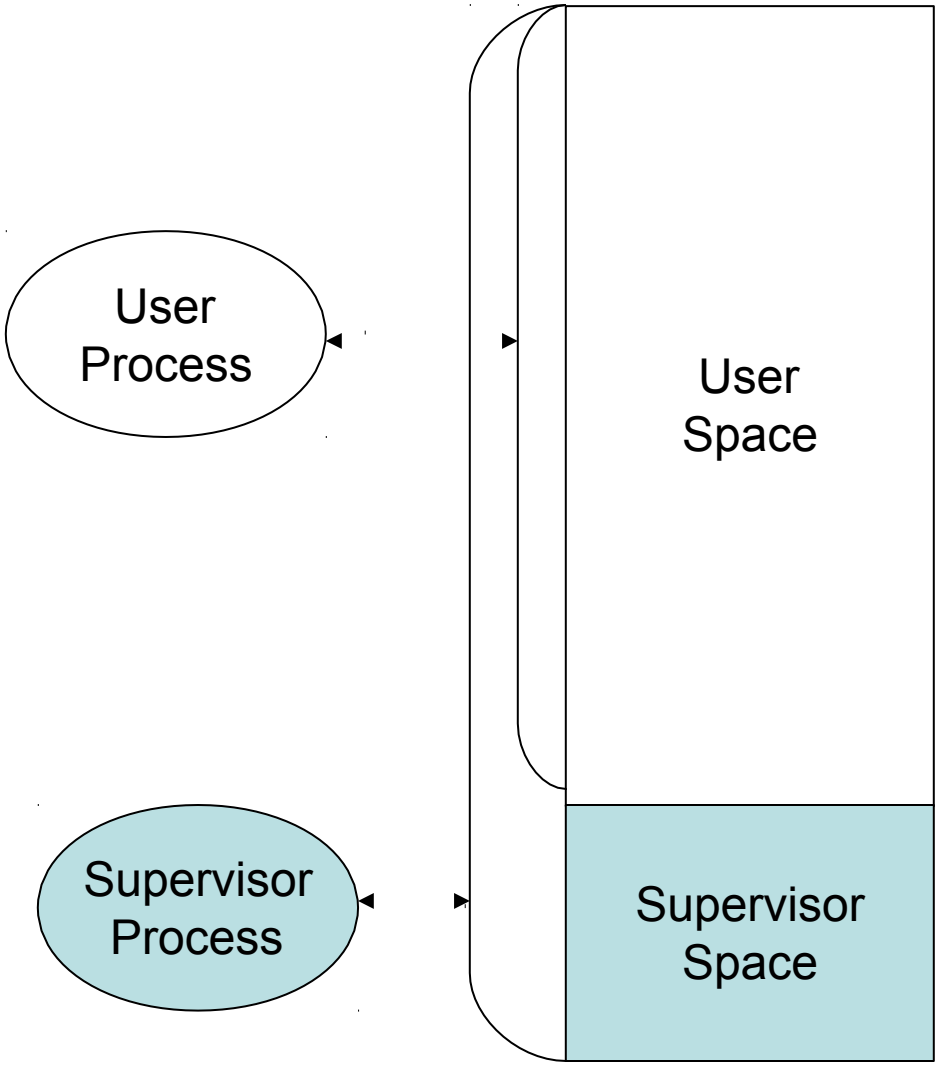


(Virtual) memory segregation

- A modern computer operating system usually segregates virtual memory into **kernel space** and **user space**.
- Memory separation provides memory protection and hardware protection from malicious behaviour.
- **Kernel space** is reserved for running an OS kernel.
- **User space** is where application software execute.

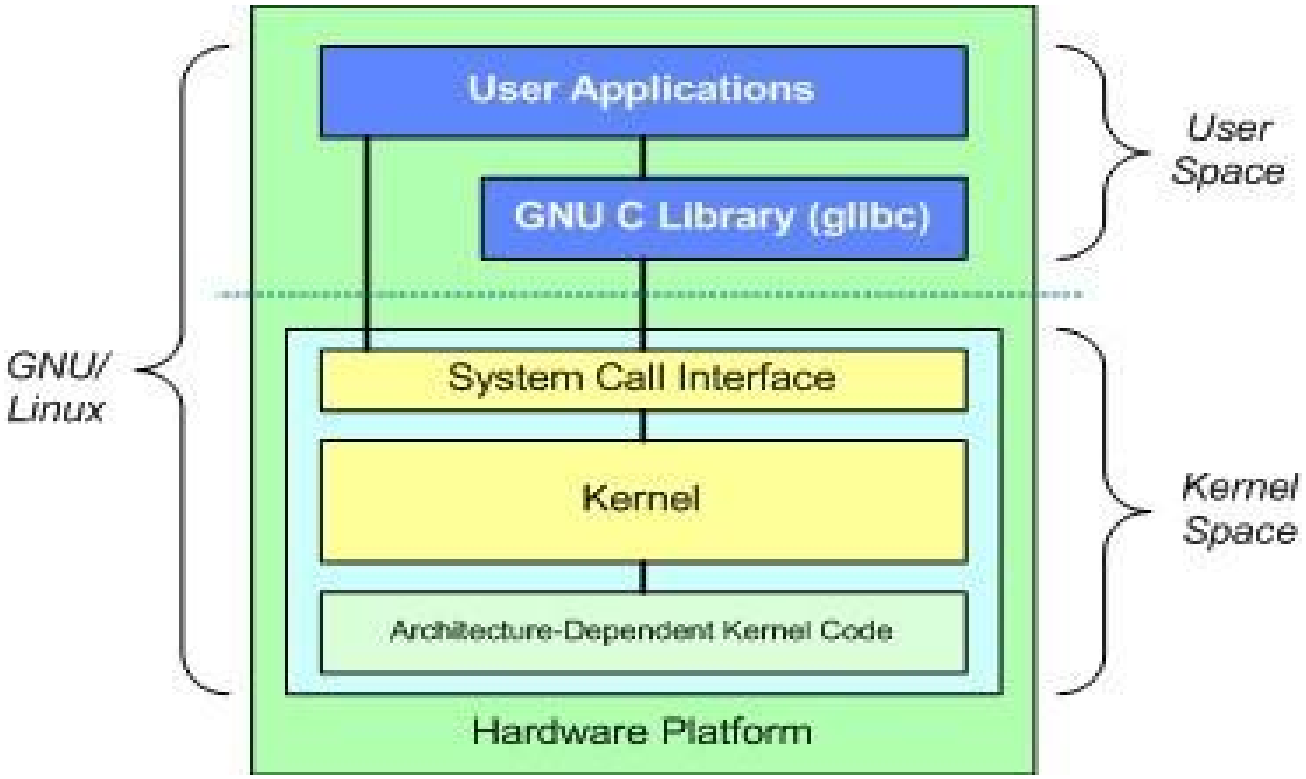
Processor Modes

- Kernel needs to make sure applications do not perform operations that harm other applications
- Give system process and user process different access using processor modes
- Mode bit may define areas of memory to be used when the processor is in supervisor mode vs user mode



Processor Modes

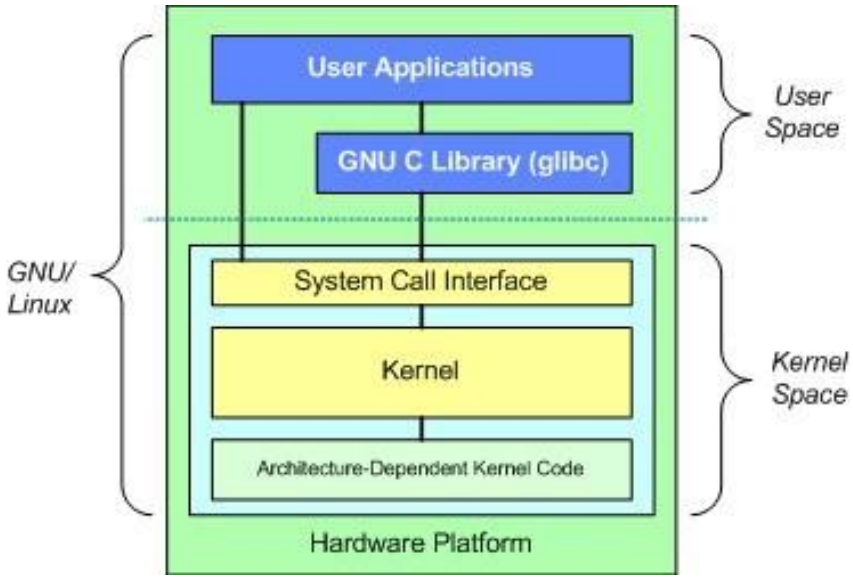
- Mode bit used to distinguish between execution on behalf of OS & behalf of user
- Supervisor mode: processor executes every instruction in it's hardware repertoire
- User mode: can only use a subset of instructions



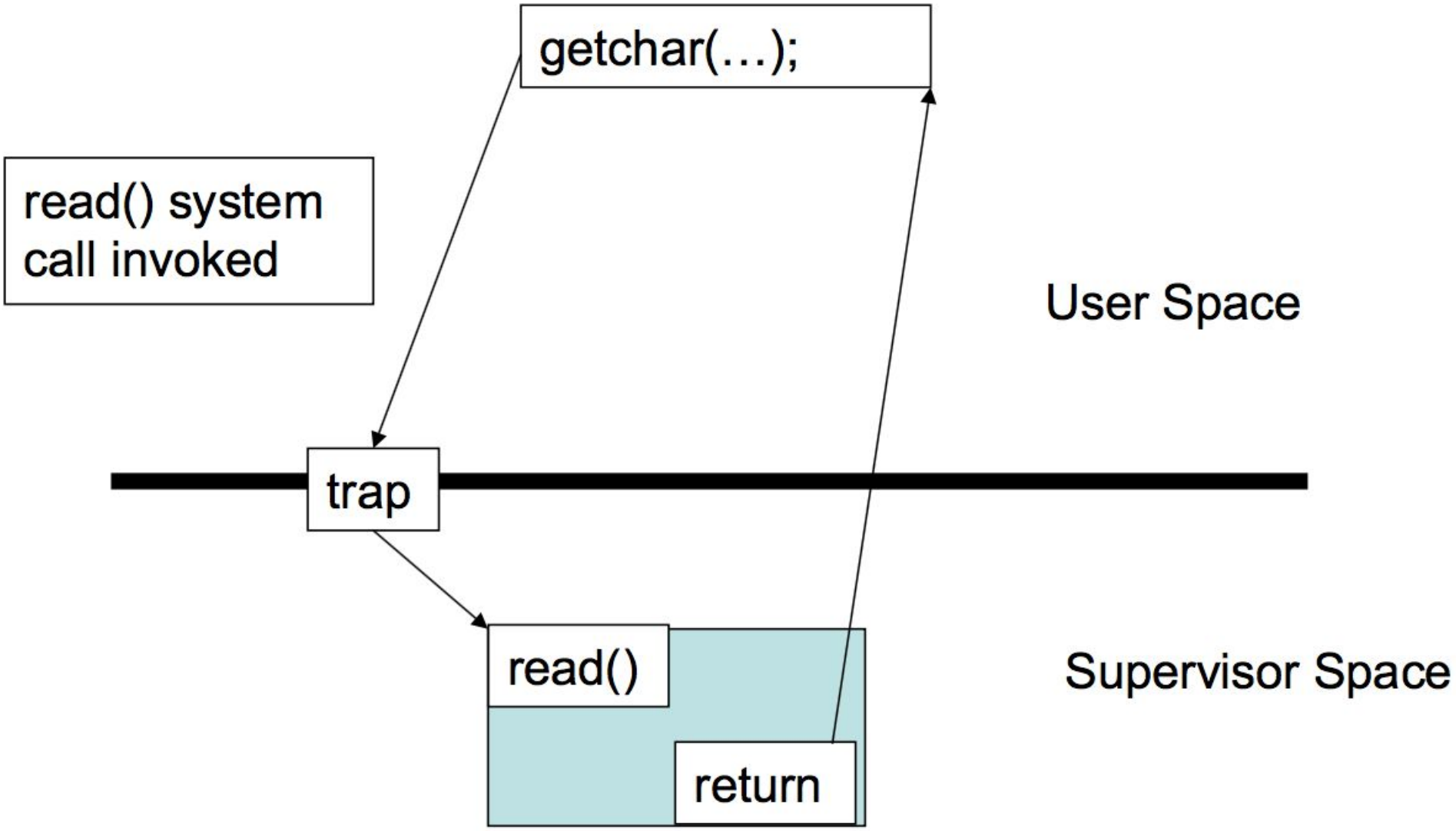
- Programs in kernel space have access to all part of memory
- Programs in user space have limited access and can only ask for certain amount of functions in kernel space via **System calls**

System call

- System calls are the interface to the kernel.
- System calls are defined by the underlying operating system and may not be fully portable
- Applications need to context switch from user space to kernel space in order to use system calls
- Overhead of system calls is high



System Calls



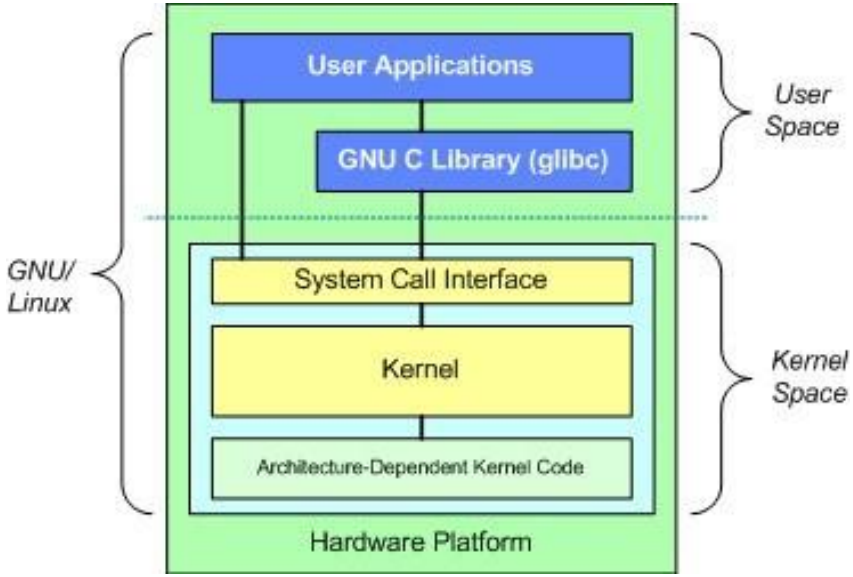
Trap: System call causes a switch from user mode to kernel mode

System calls

- A system call involves the following
  - The system call causes a ‘trap’ that interrupts the execution of the user process (user mode)
  - The kernel takes control of the processor(kernel mode\privilege switch)
  - The kernel executes the system call on behalf of the user process
  - The user process gets back control of the processor (user mode\privilege switch)
- System calls have to be used **with care**.
- Expensive due to **privilege switching**

Library functions

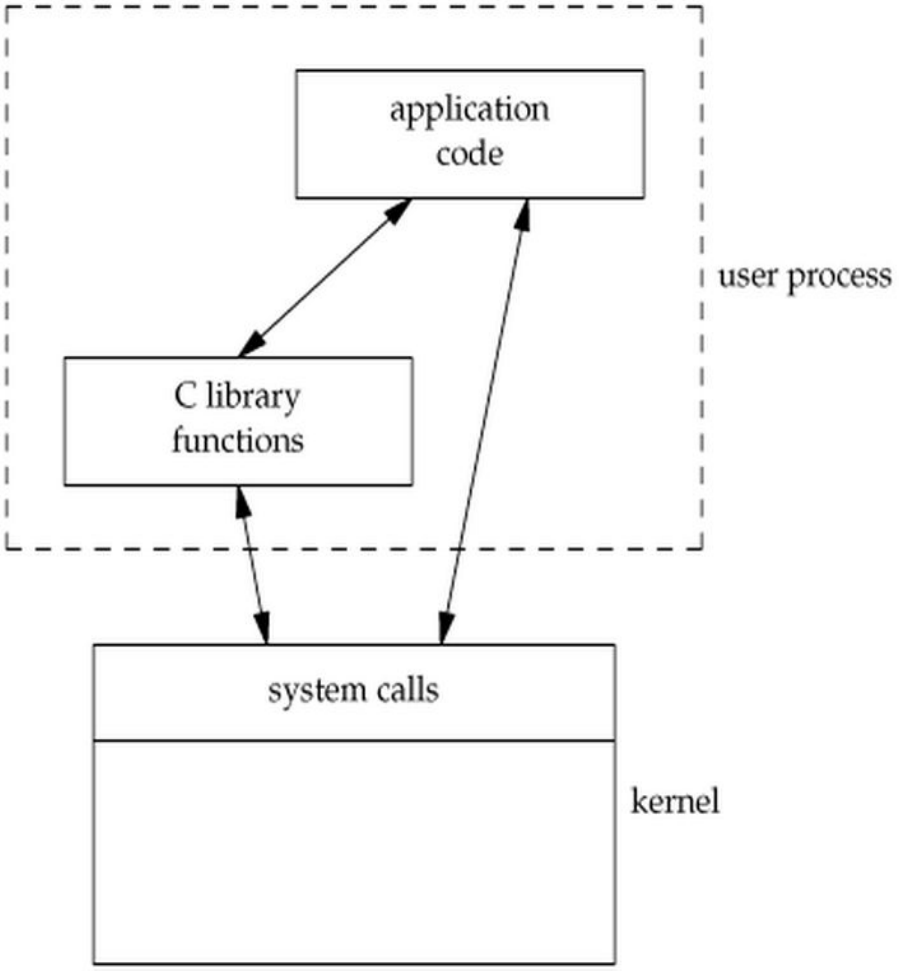
- Library functions are set of functions that can be used by application
- Library functions usually adhere to certain standard (i.e. ANSI C standard library)
- Library functions usually are usually executed in user space
- No context switch, less overhead



Library Functions

- To avoid system call overhead use equivalent library functions
  - getchar, putchar vs. read, write (for standard I/O)
  - fopen, fclose vs. open, close (for file I/O), etc.
- How do these functions perform privileged operations?
  - They make system calls

So What's the Point?



- Many library functions invoke system calls indirectly
- So why use library calls?
- Usually equivalent library functions make fewer system calls
- non-frequent switches from user mode to kernel mode => less overhead

Unbuffered vs. Buffered I/O

- **Unbuffered**
  - Every byte is read/written by the kernel through a system call
- **Buffered**
  - collect as many bytes as possible (in a buffer) and read more than a single byte (into buffer) at a time and use one system call for a block of bytes

=> Buffered I/O decreases the number of read/write system calls and the corresponding overhead

Unbuffered vs. Buffered I/O examples

- **Buffered** output improves I/O performance and can reduce system calls.
- **Unbuffered** output when you want to ensure that the output has been written before continuing.
  - **stderr** under a C runtime library is unbuffered by default. Errors are infrequent, but want to know about them immediately.
  - **stdout** *is* buffered because it's assumed there will be far more data going through it.
  - **logging**: log messages of a process?

Buffering issues

- What is buffering?
- Why do we buffer?
- Can we make our buffer really big?

Hints for Assignment 6

Cite Jin Wang

Lab 6: requirements

- Programs tr2b and tr2u in 'C‘:
  - Take two arguments 'from' and 'to'.
  - Transliterate every **byte** in 'from' to corresponding byte in 'to‘
  - e.g. Replace 'a' with 'w', 'b' with 'x’:
    ./tr2b 'abcd' 'wxyz' < bigfile.txt
- tr2b: uses **getchar/putchar,** read from STDIN and write to STDOUT
- tr2u: uses **read/write** to read and write each byte
  - The nbyte argument should be 1

Lab 6: hints

- Test it on a big file with 5000000 bytes

  generate big file: for i = 1 to 5,000,000 .....
- Compare system calls
  - Use command *strace -c*
- Test the running time
  - Use command *time*

Homework 6

- Recall Homework 5!
- Rewrite *sfrob* using system calls (*sfrobu*)
- *sfrobu* should behave like *sfrob* except
  - If stdin is a regular file, it should initially allocate enough memory to **hold all data in the file all at once**
  - It outputs a line with the number of comparisons performed
- System call functions you'll need: **read, write, and fstat**

Homework 6

- Measure differences in performance between *sfrob* and *sfrobu* using the time command
- Estimate the number of comparisons as a function of the number of input lines provided to *sfrobu*
- Write a shell script "*sfrobs*" that uses tr and the sort utility to perform the same overall operation as *sfrob*
- Encrypted input -> tr (decrypt) -> sort (sort decrypted text) -> tr (encrypt) -> encrypted output

System calls

- `ssize_t read(int fildes, void *buf, size_t nbyte)`
  - fildes: file descriptor
  - buf: buffer to write to
  - nbyte: number of bytes to read
- `ssize_t write(int fildes,const void *buf,size_t nbyte)`
  - fildes: file descriptor
  - buf: buffer to write to
  - nbyte: number of bytes to write
- `int open(const char *pathname,int flags,mode_t mode)`
- `int close(int fd)`
- File descriptors:
  - 0 stdin
  - 1 stdout
  - 2 stderr
- *Why are these system calls and not just regular library functions?*

- `pid_t getpid(void)`
  - returns the process id of the calling process
- `int dup(int fd)`
  - Duplicates a file descriptor fd. Returns a second file descriptor that points to the same file table entry as fd does.
- `int fstat(int filedes, struct stat *buf)`
  - Returns information about the file with the descriptor filedes to buf

More examples: System calls

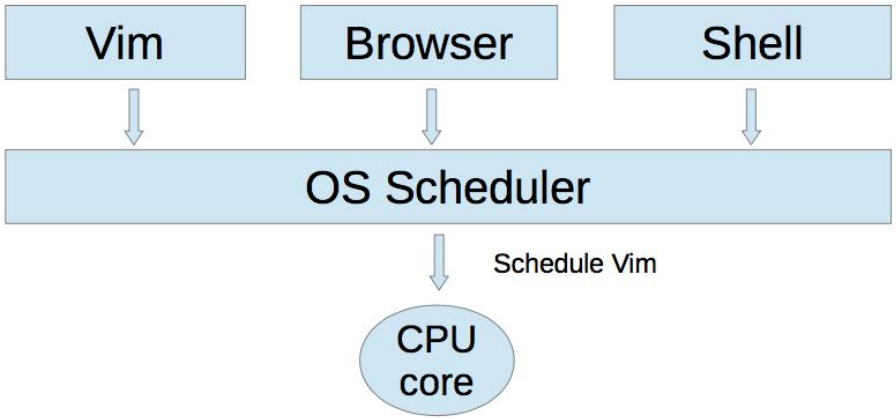
```
struct stat {
dev_t     st_dev;         /* ID of device containing file */
ino_t     st_ino;         /* inode number */
mode_t    st_mode;        /* protection */
nlink_t   st_nlink;       /* number of hard links */
uid_t     st_uid;         /* user ID of owner */
gid_t     st_gid;         /* group ID of owner */
dev_t     st_rdev;        /* device ID (if special file) */
off_t     st_size;        /* total size, in bytes */
blksize_t st_blksize;     /* blocksize for filesystem I/O */
blkcnt_t  st_blocks;      /* number of 512B blocks allocated */

time_t st_atime;  /* time of last access */
time_t st_mtime;  /* time of last modification */
time_t st_ctime;  /* time of last status change */
};
```

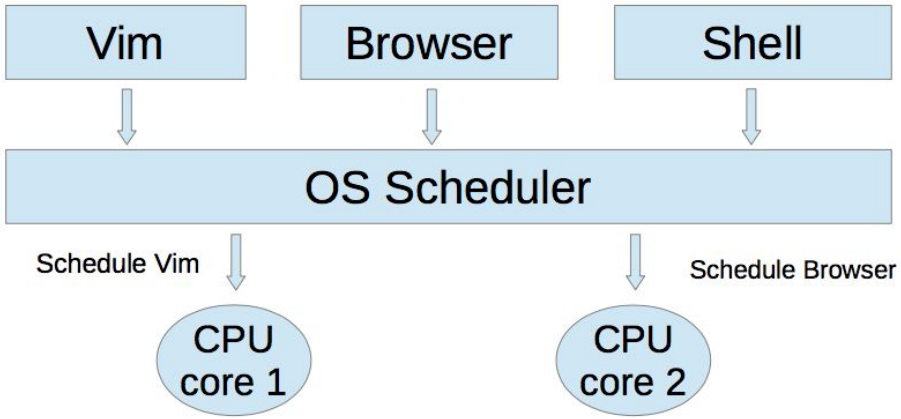
Multithreading/Parallel Processing

Week 6

Multitasking

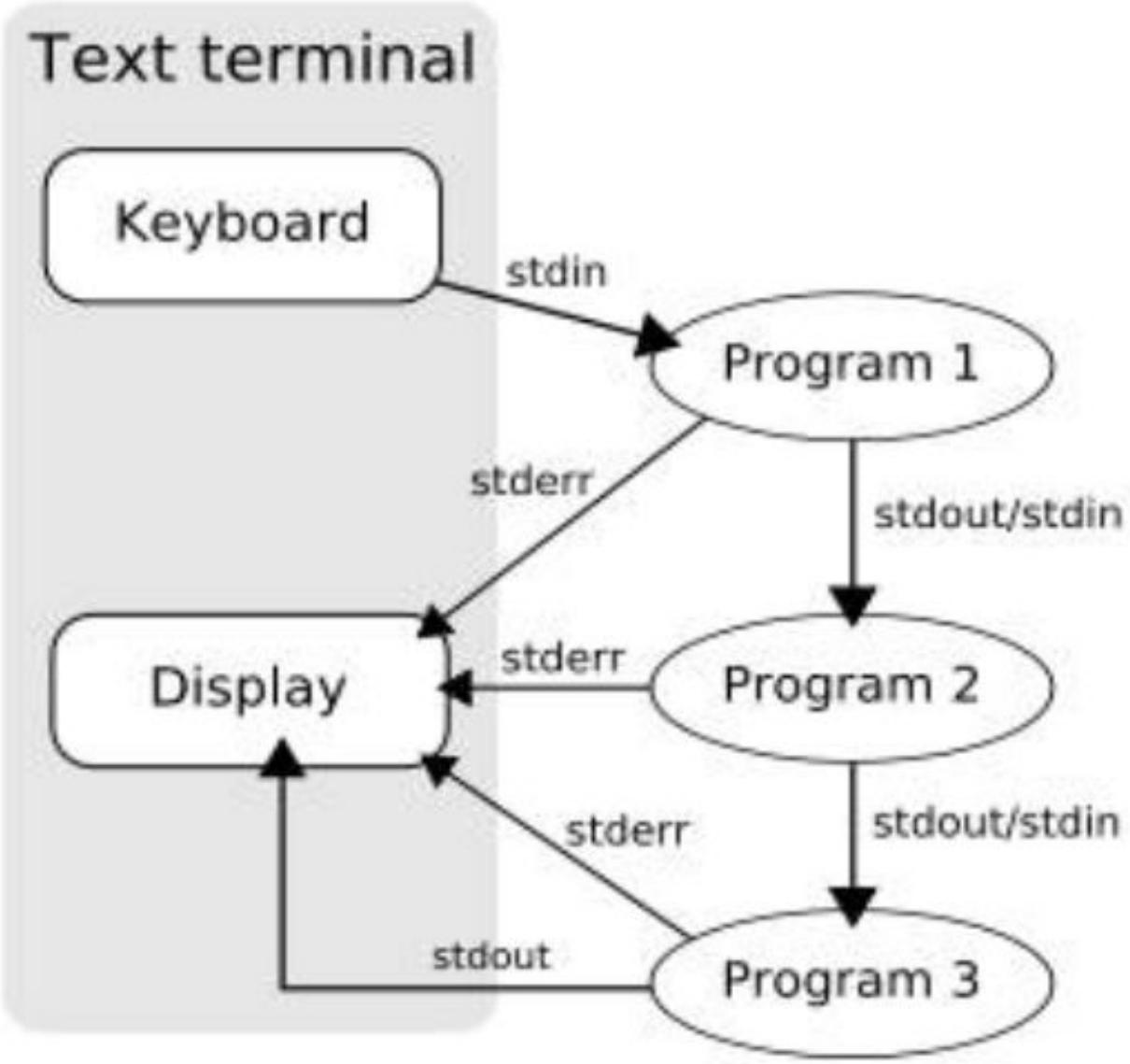
- Run multiple processes **simultaneously** to increase performance
- Processes do not share internal structures (stacks,globals,etc)
  - Communicate via **IPC** (inter-process communication) methods
    - Pipes, sockets, signals, message queues
- **Single core: Illusion** of parallelism by switching processes quickly (**time-sharing**). Why is illusion good?
- **Multi-core: True** parallelism. Multiple processes execute **concurrently** on different CPU cores


Vim
Browser
Shell
OS Scheduler
Schedule Vim
CPU
core


Vim
Browser
Shell
OS Scheduler
Schedule Vim
Schedule Browser
CPU
core 1
CPU
core 2

Multitasking

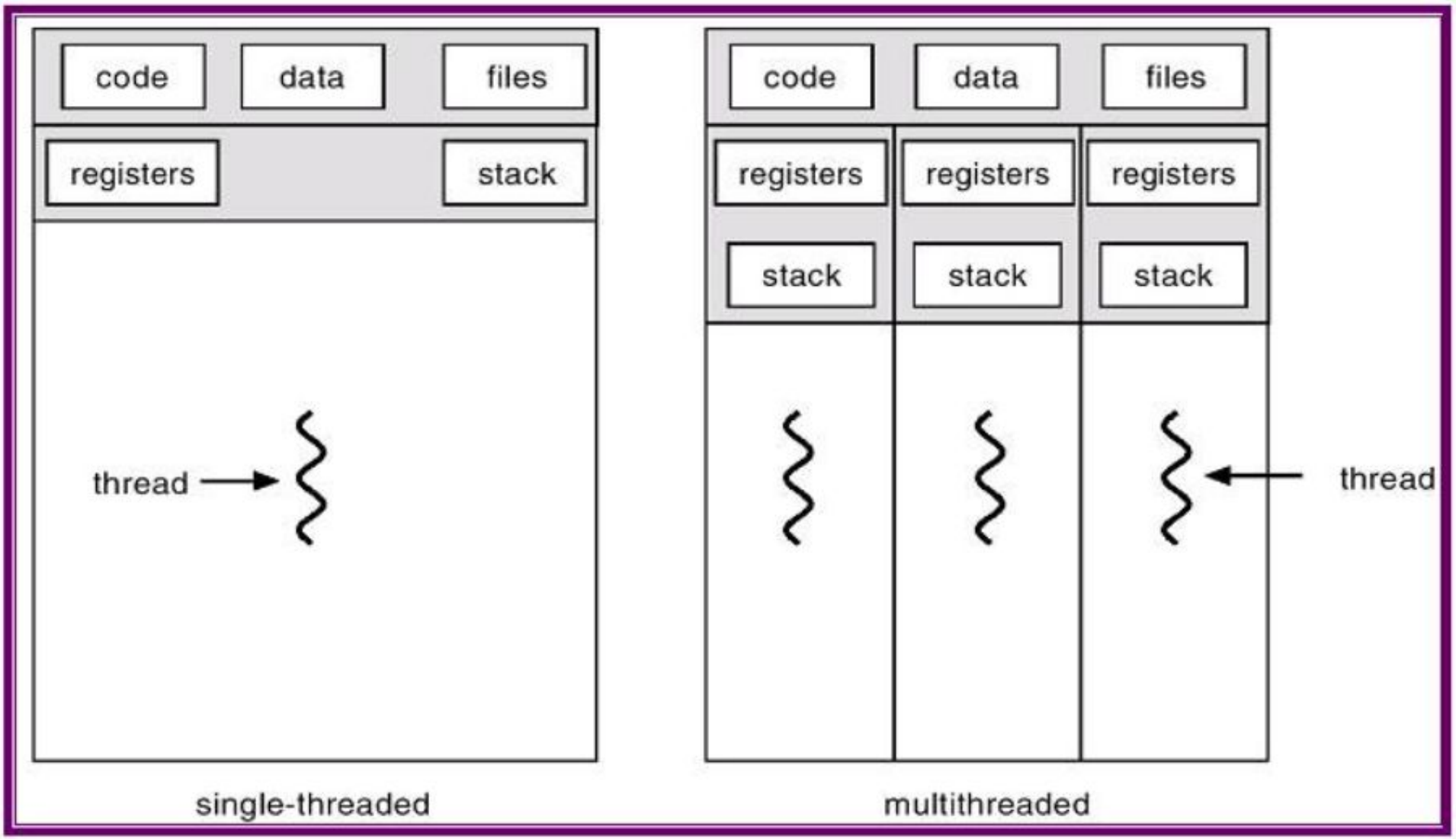
- tr -s ‘[:space:]’ ‘\n’ | sort -u | comm -23 - words
- Three separate processes spawned simultaneously
  - P1 - tr
  - P2 - sort
  - P3 - comm
- Common buffers (pipes) exist between 2 processes for communication
  - ‘tr’ writes its stdout to a buffer that is read by ‘sort’
  - ‘sort’ can execute, as and when data is available in the buffer
  - Similarly, a buffer is used for communicating between ‘sort’ and ‘comm


Text terminal
Keyboard
stdin
Program 1
stderr
stdout/stdin
Display
stderr
Program 2
stderr
stdout/stdin
stdout
Program 3

Threads

- A flow of instructions, path of execution within a process
- It is a basic unit of CPU utilization
- Each thread has its own:
  - Stack
  - Registers
  - Thread ID
- Each thread shares the following with other threads belonging to the same process
  - Code
  - Heap
  - Global Data
  - OS resources (files,I/O)
- A process can be single-threaded or multi-threaded
- Threads in a process can run in parallel

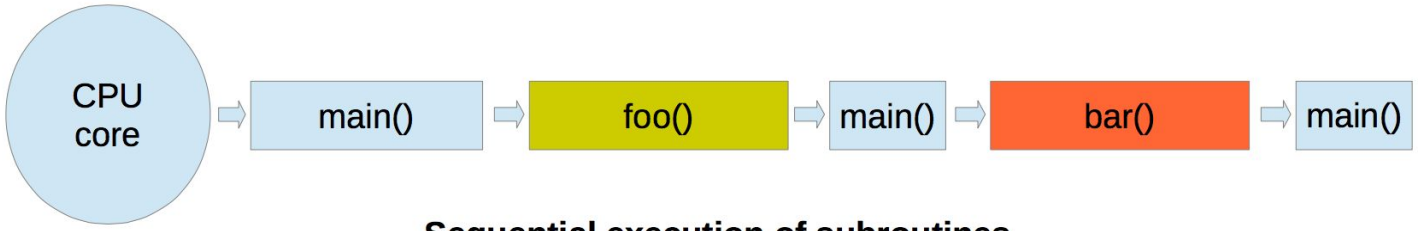
  (provide another type of parallelism)


code
data
files
registers
stack
thread
single-threaded
code
data
files
registers
registers
registers
stack
stack
stack
thread
multithreaded

Single threaded execution

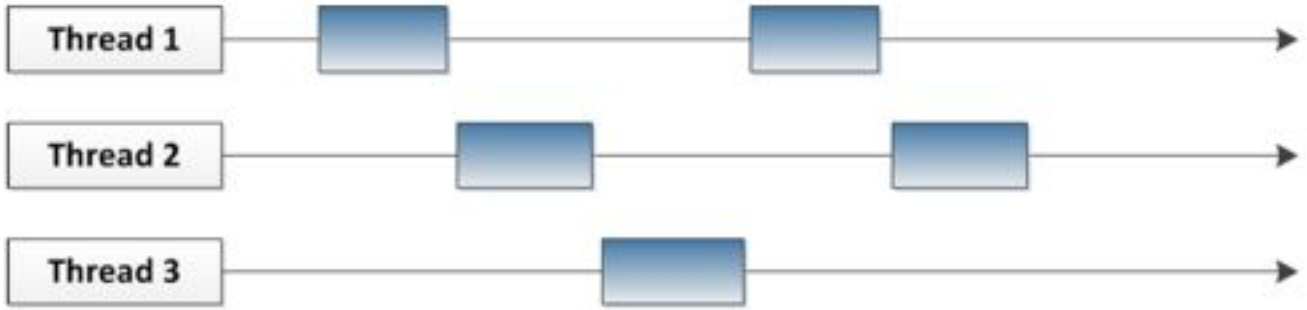
```
int global_counter = 0
int main()
{
        …
        foo(arg1,arg2);
        bar(arg3,arg4,arg5);
        …
        return 0;
}
```

```
void foo(arg1,arg2)
{
        //code for foo
}
void bar(arg3,arg4,arg5)
{
        //code for bar
}
```



Sequential execution of subroutines

Multiple threads sharing a single CPU

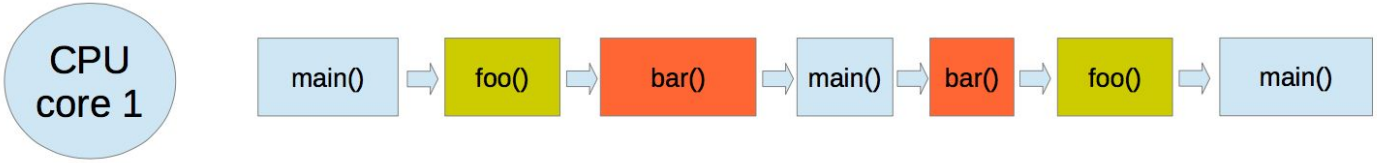


Multiple threads on multiple CPUs

Multi threaded execution (single core)

```
int global_counter = 0
int main()
{
        …
        foo(arg1,arg2);
        bar(arg3,arg4,arg5);
        …
        return 0;
}
```

```
void foo(arg1,arg2)
{
        //code for foo
}
void bar(arg3,arg4,arg5)
{
        //code for bar
}
```

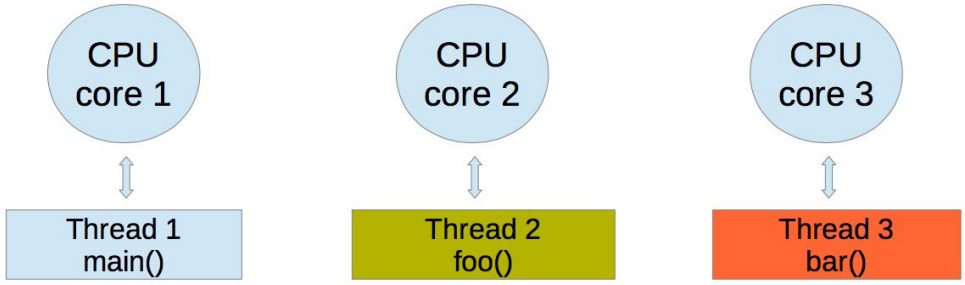


Time Sharing – Illusion of multithreaded parallelism
(Thread switching has less overhead compared to process switching)

Multi threaded execution (multiple cores)

```
int global_counter = 0
int main()
{
        …
        foo(arg1,arg2);
        bar(arg3,arg4,arg5);
        …
        return 0;
}
```

```
void foo(arg1,arg2)
{
        //code for foo
}
void bar(arg3,arg4,arg5)
{
        //code for bar
}
```



True multithreaded parallelism

Multithreading properties

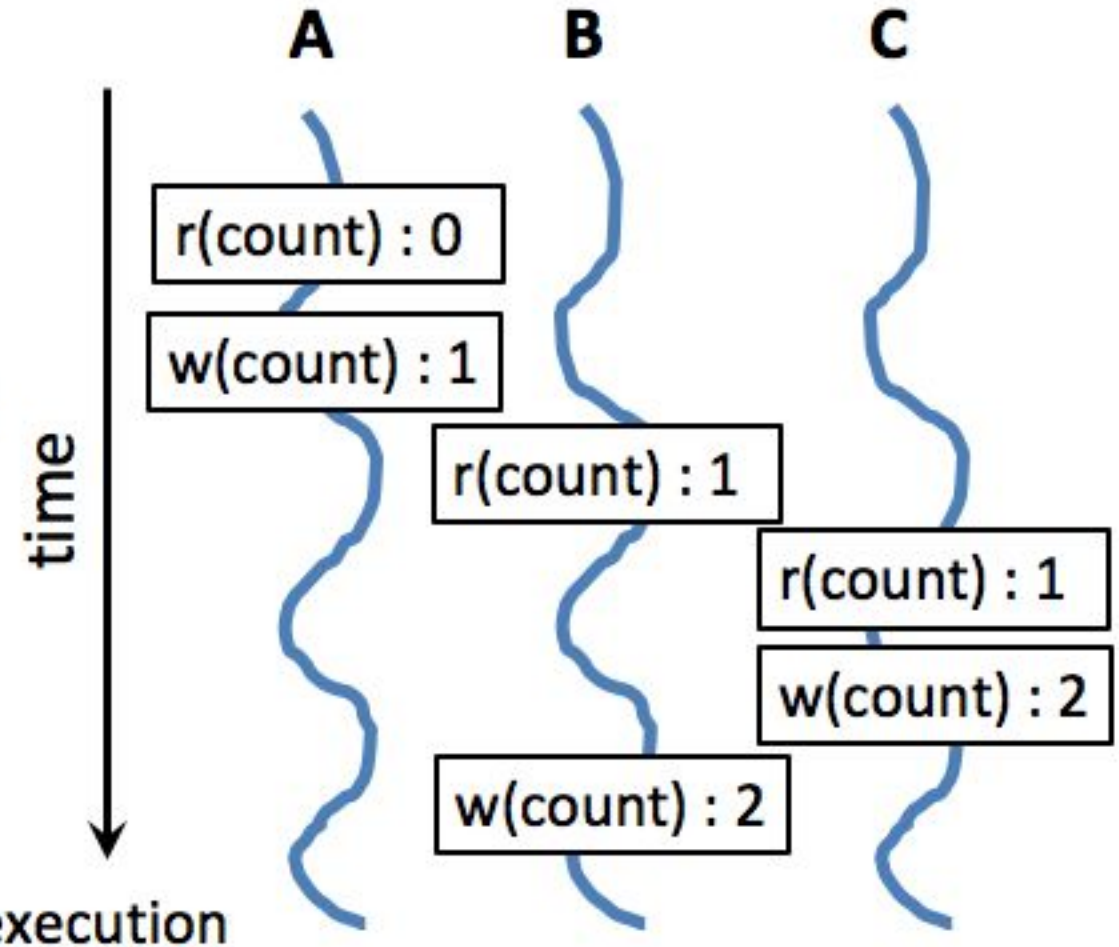
- Efficient way to **parallelize** tasks
- **Thread switches are less expensive** compared to process switches (context switching)
- Inter-thread communication is easy, via **shared global** data (heap)
- Need **synchronization** among threads accessing same data

Shared Memory

- Makes multithreaded programming
  - Powerful
    can easily access data and share it among threads
  - More efficient
    No need for system calls when sharing data
    Thread creation and destruction less expensive than process creation and destruction
  - Non-trivial
    Have to prevent several threads from accessing and changing the same shared data at the same time (synchronization)

Race Condition

```
int count = 0;
void increment()
{
  count = count + 1;
}
```



Result depends on order of execution

=> Synchronization needed

Lab 6

- Evaluate the performance of multithreaded ‘sort’ command
  - **od -An -f -N 4000000 < /dev/urandom | tr -s ' ' '\n' > random.txt**
  - Might have to modify the command above
- Delete the empty line
  - **time -p sort -g --parallel=2 numbers.txt > /dev/null**
- Add /usr/local/cs/bin to PATH
  - $ export PATH=/usr/local/cs/bin:$PATH
- Generate a file containing 10M random **single-precision floating point numbers**, one per line with no white space
  - /dev/urandom: pseudo-random number generator

Lab 6

- od
  - write the contents of its input files to standard output in a user-specified format
  - Options
    - -t f: Double-precision floating point
    - -N <count>: Format no more than *count* bytes of input
- sed, tr
  - Remove address, delete spaces, add newlines between each float

Lab 6

- use time -p to time the command sort -g on the data you generated
- Send output to /dev/null
- Run sort with the --parallel option and the
  - –g option: compare by general numeric value
  - Use time command to record the real, user and system time when running sort with 1, 2, 4, and 8 threads
    - $ time –p sort –g file_name > /dev/null (1 thread)
    - $ time –p sort –g --parallel=[2, 4, or 8] file_name > /dev/null
  - Record the times and steps in log.txt

CS 35L Software Construction Lab
Week 6 – Multithreading

Threads

- A flow of instructions, path of execution within a process
- It is a basic unit of CPU utilization
- Each thread has its own:
  - Stack
  - Registers
  - Thread ID
- Each thread shares the following with other threads belonging to the same process
  - Code
  - Heap
  - Global Data
  - OS resources (files,I/O)
- A process can be single-threaded or multi-threaded
- Threads in a process can run in parallel

  (provide another type of parallelism)

Thread safety/synchronization

- **Thread safe function** - safe to be called by multiple threads at the same time. Function is free of 'race conditions' when called by multiple threads simultaneously.
- **Race condition** - the output depends on the order of execution
  - Shared data changed by 2 threads
    - int balance = 1000
  - Thread 1
    - T1 - read balance
    - T1 - Deduct 50 from balance
    - T1 - update balance with new value
  - Thread 2
    - T2 - read balance
    - T2 - add 150 to balance
    - T2 - update balance with new value

Thread safety/synchronization

- Order 1
  - balance = 1000
  - T1 - Read balance (1000)
  - T1 - Deduct 50
    - 950 in temporary result
  - T2 - read balance (1000)
  - T1 - update balance
    - balance is 950 at this point
  - T2 - add 150 to balance
    - 1150 in temporary result
  - T2 - update balance
    - balance is 1150 at this point
  - **The final value of balance is 1150**
- Order 2
  - balance = 1000
  - T1 - read balance (1000)
  - T2 - read balance (1000)
  - T2 - add 150 to balance
    - 1150 in temporary result
  - T1 - Deduct 50
    - 950 in temporary result
  - T2 - update balance
    - balance is 1150 at this point
  - T1 - update balance
    - balance is 950 at this point
  - **The final value of balance is 950**

Thread synchronization

- Only one thread will get the mutex. Other thread will **block in Mutex.lock()**
- Other thread can start execution only when the thread that holds the mutex calls **Mutex.unlock()**

Thread synchronization

- **Mutex (mutual exclusion)**
  - Thread 1
    - Mutex.lock()
      - Read balance
      - Deduct 50 from balance
      - Update balance with new value
    - Mutex.unlock()
  - Thread 2
    - Mutex.lock()
      - Read balance
      - Add 150 to balance
      - Update balance with new value
    - Mutex.unlock()
  - balance = 1100

Pthread in C

- compile with -pthread (or -lpthread) compiler option

-tells the compiler that your program requires threading support

- include pthread.h in program header

Basic pthread Functions

There are 5 basic pthread functions:

1. **pthread_create:** creates a new thread within a process

2. **pthread_join:** waits for another thread to terminate

3. **pthread_equal:** compares thread ids to see if they refer to the same thread

4. **pthread_self:** returns the id of the calling thread

5. **pthread_exit:** terminates the currently running thread

pthread_create

- **Function:** starts a new thread in the calling process
- Return value:
  - Success: zero
  - Failure: error number

Parameters

int pthread_create(pthread_t *tid, const pthread_attr_t *attr, void *(my_function)(void *), void *arg);

- **tid**: unique identifier for newly created thread
- **attr**: object that holds thread attributes (priority, stack size, etc.)
  - Pass in NULL for default attributes
- **my_function**: function that thread will execute once it is created
- **arg**: a *single* argument that may be passed to my_function
  - Pass in NULL if no arguments

pthread_create Example

```
#include <pthread.h> …
void *printMsg(void *thread_num) {
        int t_num = (int) thread_num;
        printf("It's me, thread #%d!\n", t_num); }

int main() {
        pthread_t tids[3];
        int t;
        for(t = 0; t < 3; t++) {
                ret = pthread_create(&tids[t], NULL, printMsg, (void *) t);
                if(ret) {
                        printf("Error creating thread. Error code is %d\n", ret");
                        exit(-1); }
        }
}
```

Possible problem with this code?

If main thread finishes before all threads finish their job -> incorrect results

pthread_join

- **Function:** makes originating thread wait for the completion of all its spawned threads' tasks
- Without join, the originating thread would exit as soon as it completes its job

  ⇒A spawned thread can get aborted even if it is in the middle of its chore
- Return value:

  ⇒Success: zero

  ⇒Failure: error number

Arguments

int pthread_join(pthread_t tid, void **status);

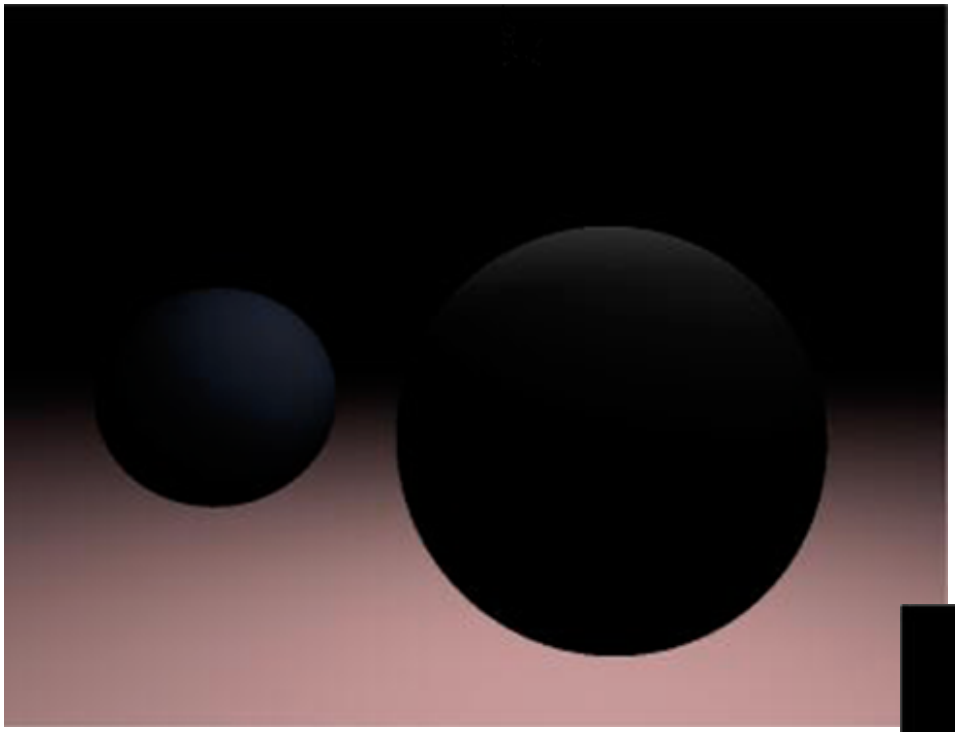
- **tid**: thread ID of thread to wait on
- **status:** the exit status of the target thread is stored in the location pointed to by *status
  - Pass in NULL if no status is needed

pthread_join Example

```
#include <pthread.h> …
#define NUM_THREADS 5

void *PrintHello(void *thread_ num) {
        printf("\n%d: Hello World!\n", (int) thread_num); }

int main() {
        pthread_t threads[NUM_THREADS];
        int ret, t;
        for(t = 0; t < NUM_THREADS; t++) {
                printf("Creating thread %d\n", t);
                ret = pthread_create(&threads[t], NULL, PrintHello, (void *) t);
                // check return value }

         for(t = 0; t < NUM_THREADS; t++) {
                ret =  pthread_join(threads[t], NULL);
                // check return value }
}
```

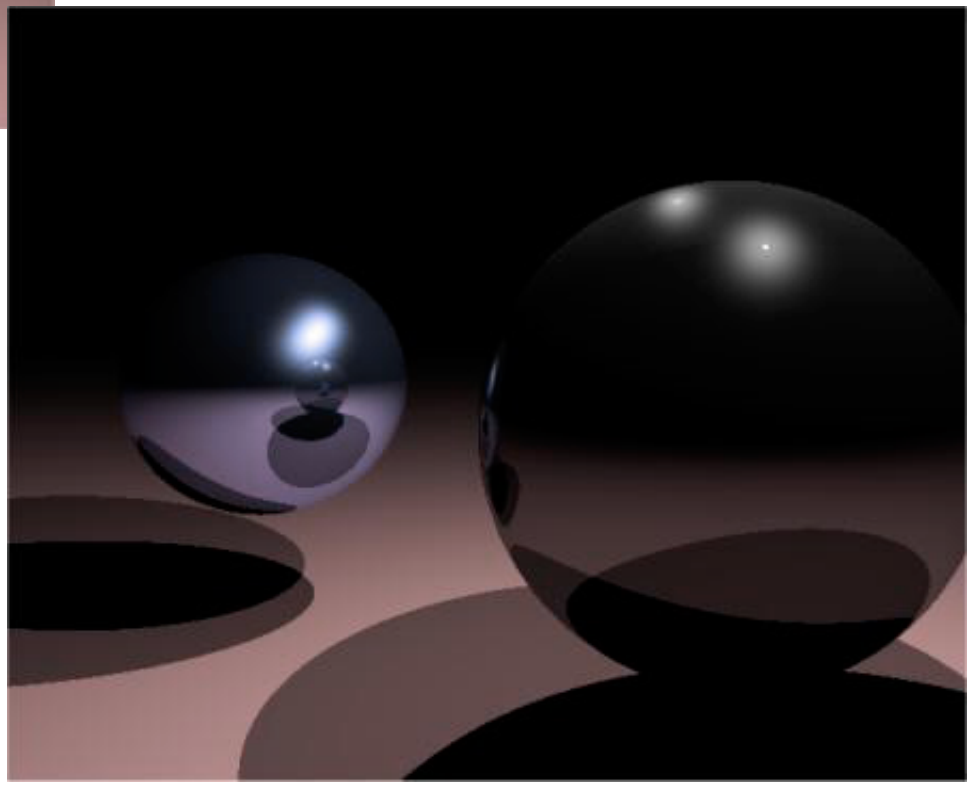
Ray Tracing

- An advanced computer graphics technique for rendering 3D images
- Mimics the propagation of light through objects
- Simulates the effects of a single light ray as it's reflected or absorbed by objects in the images



Without ray tracing

With ray tracing



Computational Resources

- Ray Tracing produces a very high degree of visual realism at a high cost
- The algorithm is *computationally intensive*

=> Good candidate for multithreading (embarrassingly parallel)

Homework 6

- Download the single-threaded ray tracer implementation
- Run it to get output image
- Multithread ray tracing
  - Modify main.c and Makefile
- Run the multithreaded version and compare resulting image with single-threaded one

Homework 6

- Build a multi-threaded version of Ray tracer
- Modify “main.c” & “Makefile”
  - Include <pthread.h> in “main.c”
  - Use “pthread_create” & “pthread_join” in “main.c”
  - Link with –lpthread flag
- make clean check
  - Outputs “1-test.ppm”
  - Can’t see “1-test.ppm”
    - sudo apt-get install gimp (Ubuntu)
    - X forwarding (lnxsrv)
    - gimp 1-test.ppm

1-test.ppm

Figure. 1-test.ppm
& baseline.ppm

SSH - Secure Shell

Week 7

Communication Over the Internet

- What type of guarantees do we want?
  - **Confidentiality**
    - Message secrecy
  - **Data integrity**
    - Message consistency
  - **Authentication**
    - Identity confirmation
  - **Authorization**
    - Specifying access rights to resources

Cryptography

- Plaintext - actual message
- Ciphertext - encrypted message (unreadable gibberish)
- Encryption - converting from plaintext to ciphertext
- Decryption - converting from ciphertext to plaintext
- Secret key
  - part of the mathematical function used to encrypt/decrypt
  - Good key makes it hard to get back plaintext from ciphertext

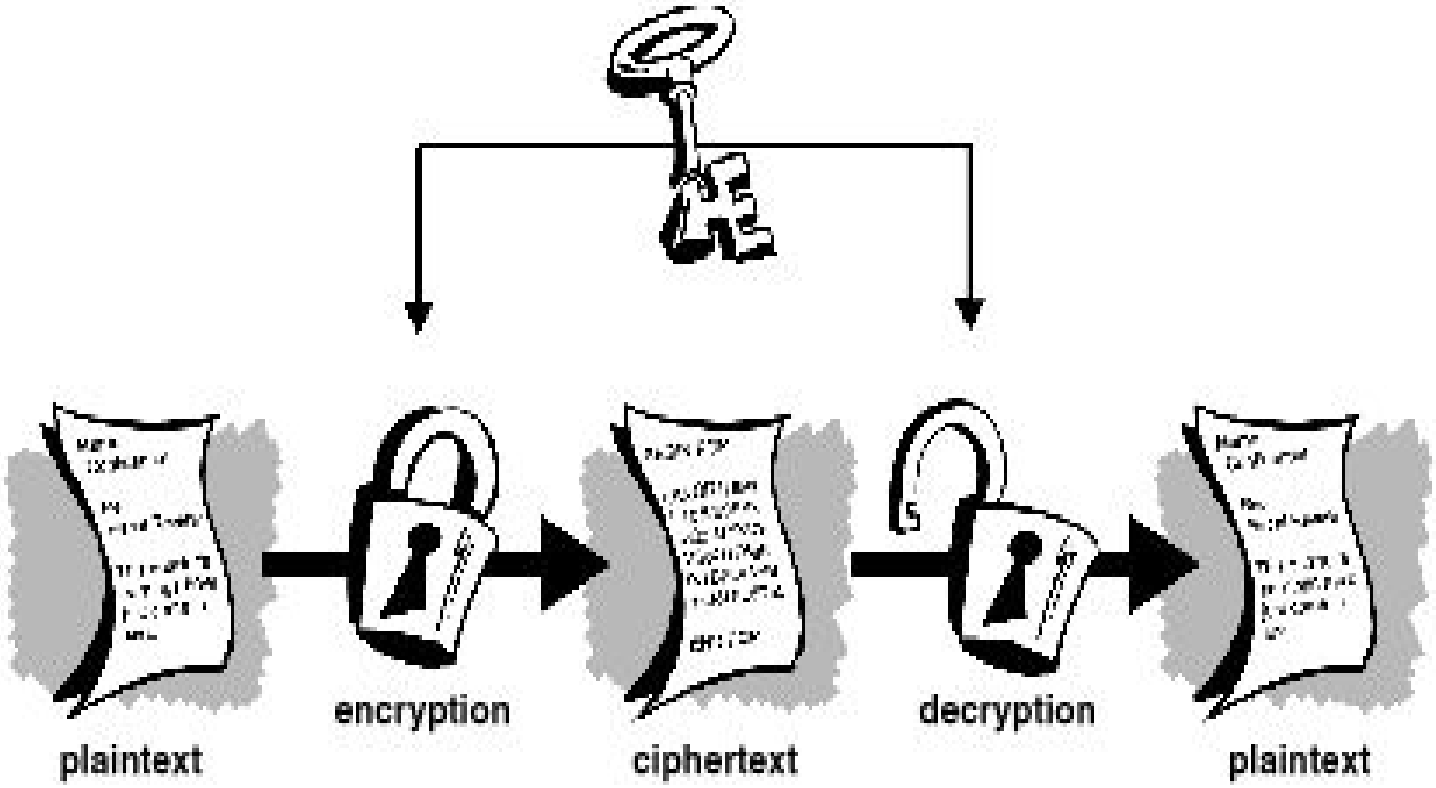


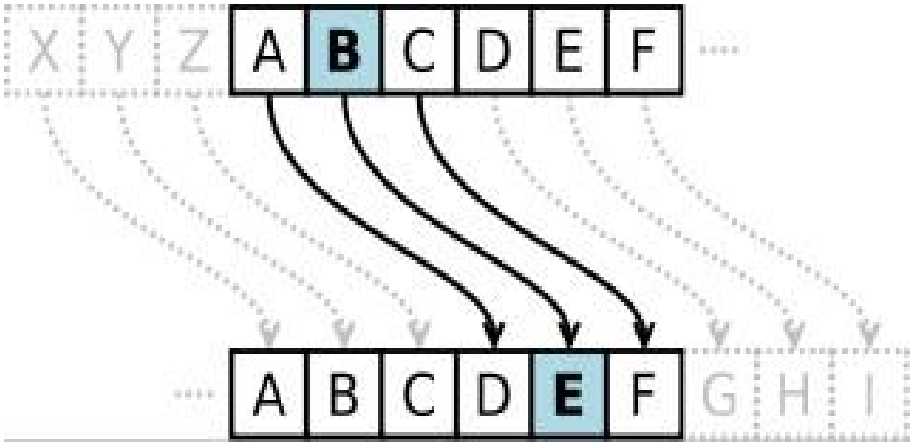
Image Source: gpgtools.org

Symmetric-key Encryption

- Same secret key used for encryption and decryption
- Example : Data Encryption Standard (DES)
- Caesar's cipher
  - Map the alphabet to a shifted version
  - Plaintext - SECRET. Ciphertext - VHFUHW
  - Key is 3 (number of shifts of the alphabet)
- Key distribution is a problem
  - The secret key has to be delivered in a safe way to the recipient
  - Chance of key being compromised

Caesar’s cipher

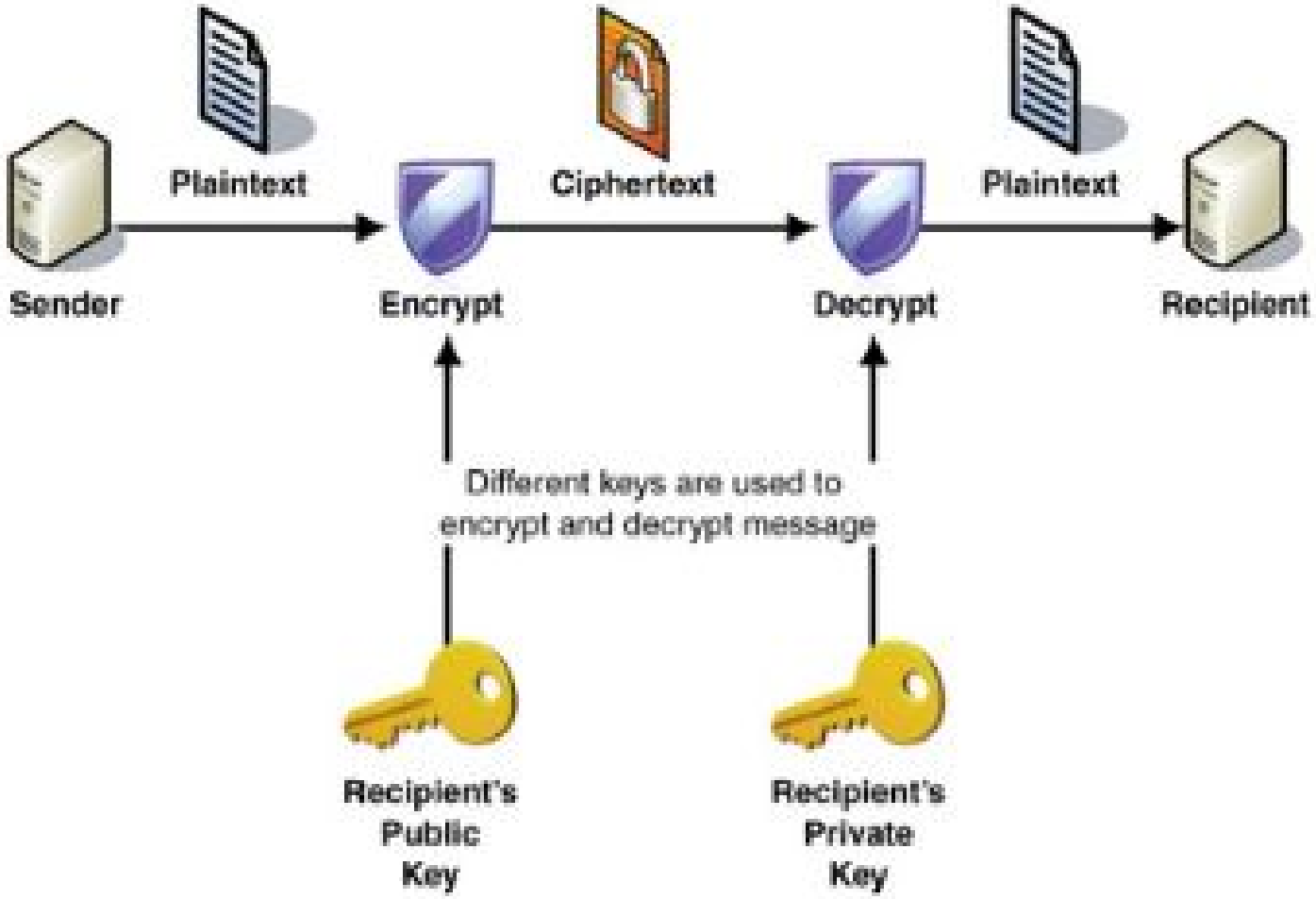
- When key is 3
  - ABCDEFGHIJKLMNOPQRSTUVWXYZ
  - DEFGHIJKLMNOPQRSTUVWXYZABC



Public-Key Encryption (Asymmetric)

- Uses a pair of keys for encryption
  - Public Key - published and well known to everyone
  - Private - Secret key known only to the owner
- Encryption
  - Use public key to encrypt messages
  - Anyone can encrypt message, but they cannot decrypt the ciphertext
- Decryption
  - Use private key to decrypt messages
- In what scheme is this encryption useful?

Public-Key Encryption (Asymmetric)



Public-Key Encryption (Asymmetric)

- Example: RSA (Rivest, Shamir & Adelman)
  - Property used: Difficulty of factoring large integers to prime numbers
  - N= p * q
  - N is a large integer.
  - p,q are prime numbers
  - N is part of the public key

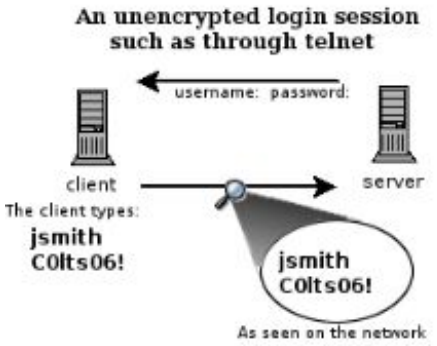
en.wikipedia.org/wiki/RSA_Factoring_Challenge

Encryption Types Comparison

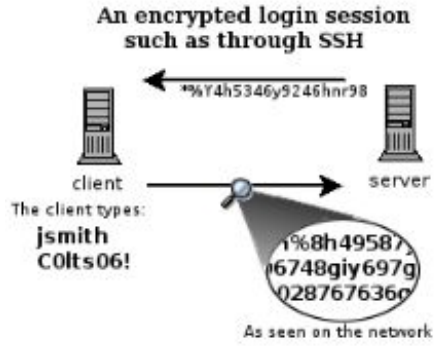
- **Symmetric Key Encryption**
  - a.k.a shared/secret key
  - Key used to encrypt is the same as key used to decrypt
- **Asymmetric Key Encryption: Public/Private**
  - 2 different (but related) keys: public and private
    - Only creator knows the relation. Private key cannot be derived from public key
  - Data encrypted with public key can only be decrypted by private key and vice versa
  - Public key can be seen by anyone
  - **Never** publish private key!!!

Secure Shell (SSH)

- Telnet
  - Remote access
  - Not encrypted
  - Packet sniffers can intercept sensitive information (username/password)

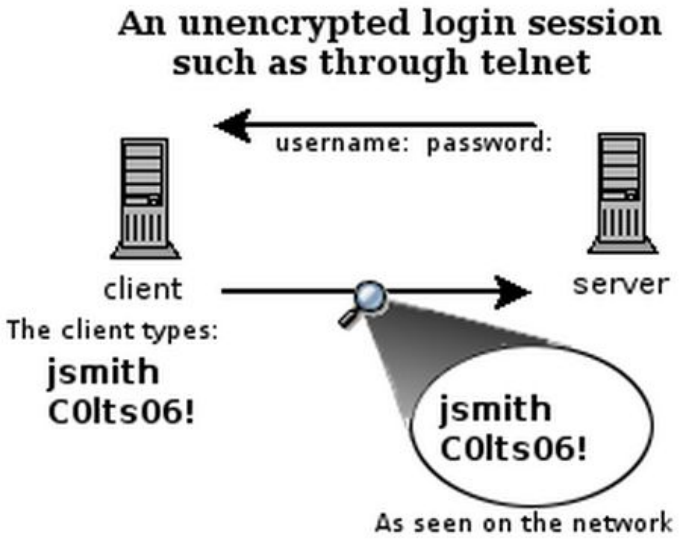


- SSH
  - run processes remotely
  - encrypted session
  - Session key (secret key) used for encryption during the session



What is SSH?

- **S**ecure **Sh**ell
- Used to remotely access shell
- Successor of telnet
- Encrypted and better authenticated session



High-Level SSH Protocol

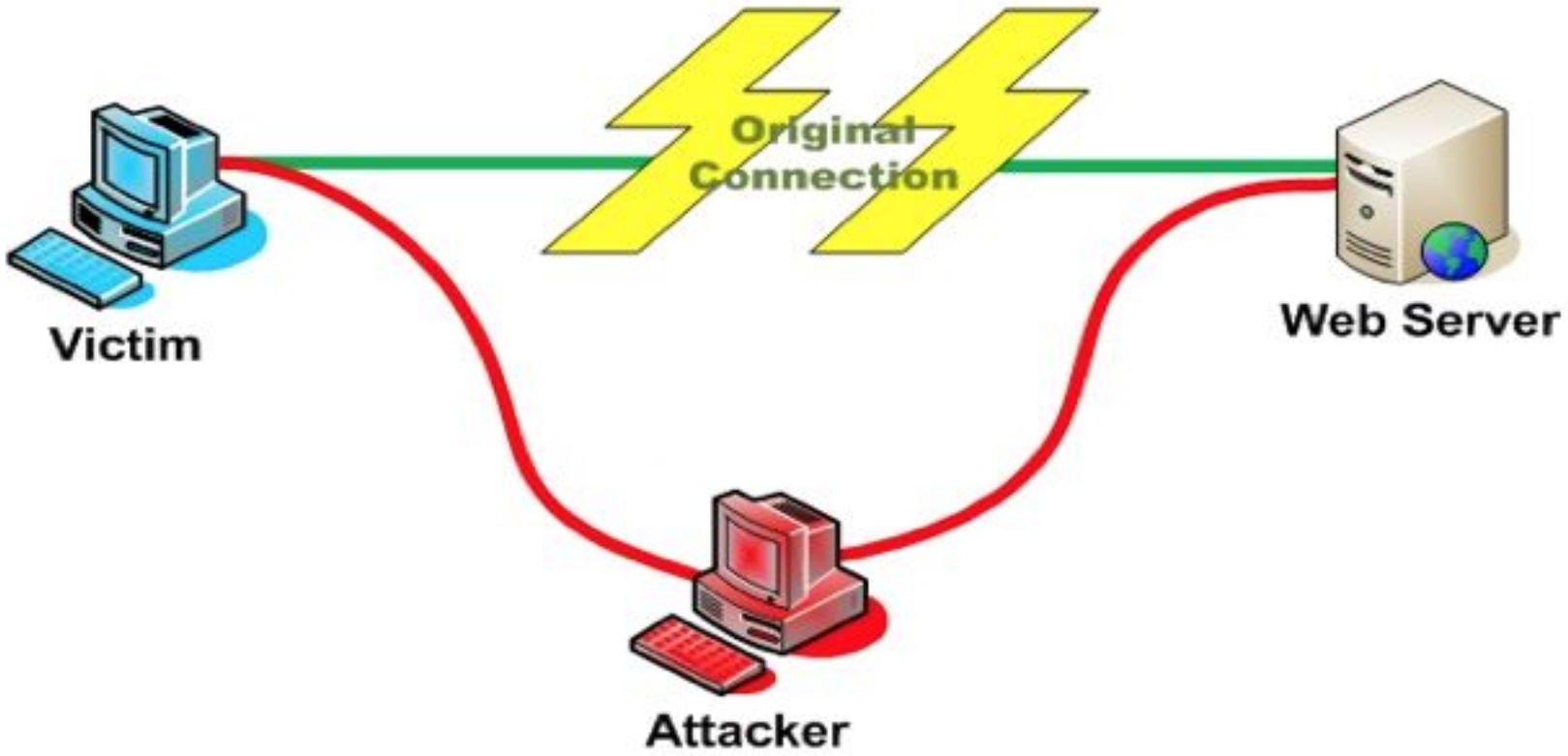
- Client ssh's to remote server
  - `$ ssh username@somehost`
  - If first time talking to server -> host validation

The authenticity of host 'somehost (192.168.1.1)' can't be established.
RSA key fingerprint is
90:9c:46:ab:03:1d:30:2c:5c:87:c5:c7:d9:13:5d:75.
Are you sure you want to continue connecting (yes/no)? **yes**
Warning: Permanently added 'somehost' (RSA) to the list of known hosts.

  - ssh doesn't know about this host yet
  - shows hostname, IP address and fingerprint of the server's public key, so you can be sure you're talking to the correct computer
  - After accepting, public key is saved in ~/.ssh/known_hosts

Host Validation

- Next time client connects to server
  - Check host's public key against saved public key
  - If they don't match



Host Validation (cont'd)

- Client asks server to prove that it is the owner of the public key using **asymmetric encryption**
  - Encrypt a message with public key
  - If server is true owner, it can decrypt the message with private key
- If everything works, host is successfully validated

User Authentication

- **Password-based authentication**
  - Prompt for password on remote server
  - If username specified exists and remote password for it is correct then the system lets you in
- **Key-based authentication**
  - Generate a key pair on the client
  - Copy the public key to the server (~/.ssh/authorized_keys)
  - Server authenticates client if it can demonstrate that it has the private key
  - The private key can be protected with a passphrase
  - Every time you ssh to a host, you will be asked for the passphrase (inconvenient!)

Secure Shell (SSH) - Client Authentication

- **Password** login
  - `ssh username@ugrad.seas.ucla.edu`
- **Passwordless** login with keys
  - Use private/public keys for authentication (server and client authentication)
  - ssh-keygen
    - Passphrase (longer version of a password/more secure)
    - Passphrase for protecting the private key
    - Passphrase needed whenever the keys are accessed
  - ssh-copy-id username@ugrad.seas.ucla.edu
    - Copies the public key to the server (~/.ssh/authorized_keys)
  - Login without password
    - ssh username@ugrad.seas.ucla.edu
    - Run scripts/commands on the remote machine
      - ssh username@ugrad.seas.ucla.edu ls
    - But you need to provide a passphrase to use a private key

ssh-agent

- A program used with OpenSSH that provides a secure way of storing the private key
- `ssh-add` prompts user for the passphrase once and adds it to the list maintained by `ssh-agent`
- Once passphrase is added to `ssh-agent`, the user will not be prompted for it again when using SSH
- OpenSSH will talk to the local ssh-agent daemon and retrieve the private key from it automatically

Secure Shell (SSH) - Client Authentication

- **Passphrase-less** authentication
  - `ssh-agent` → authentication agent
  - Manages private key identities for SSH
  - To avoid entering the passphrase whenever the key is used
  - `ssh-add`
    - Registers the private key with the agent
    - Passphrase asked only once
    - `ssh` will ask the `ssh-agent` whenever the private keys are needed

Session Encryption

- Client and server agree on a **symmetric encryption key** (session key)
- All messages sent between client and server
  - encrypted at the sender with session key
  - decrypted at the receiver with session key
- anybody who doesn't know the session key (hopefully, no one but client and server) doesn't know any of the contents of those messages

Secure Shell (SSH)

- **Session Encryption**
  - Symmetric encryption
  - Exchange secret key (Example - Diffie-Hellman)
- **Host/Client Validation**
  - Public-key Encryption
  - Challenge-Response
    - Host sends a “challenge” that has to be answered by the client
    - Similarly, client sends a “challenge” that has to be answered by the host

X session forwarding

- **X** is the windowing system for GUI apps on linux
- **X** is a network-based system. It is based upon a network protocol such that a program can run on one computer but be displayed on another
  - i.e. you want to run such apps remotely, but the GUI should show up on the local machine
- Windowing system forms the basis for most GUIs on UNIX
  - ssh -X username@ugrad.seas.ucla.edu
  - gedit
  - gimp

Secure copy (scp)

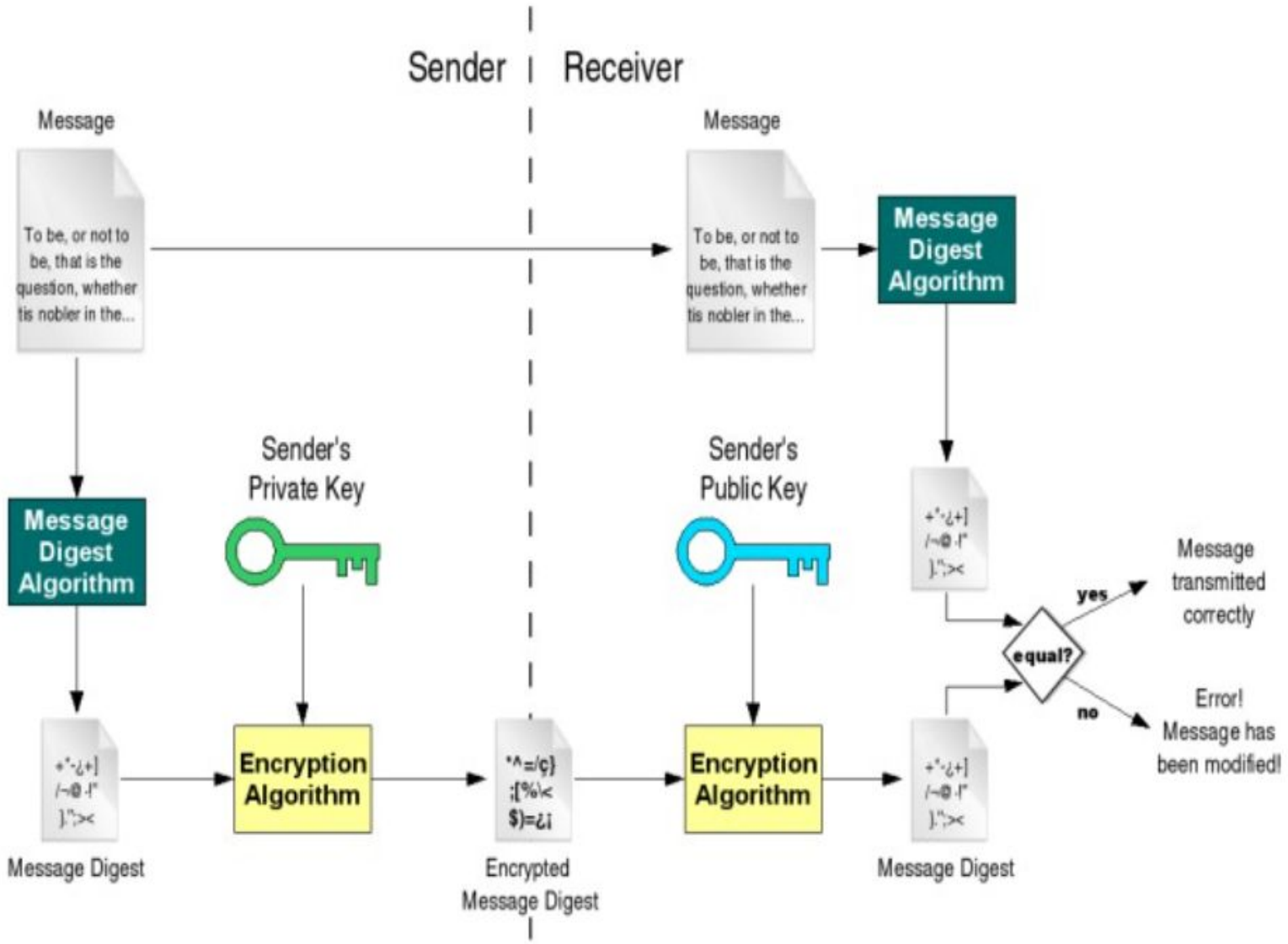
- Based on secure shell (ssh)
- Used for transferring files between hosts in a secure way (encrypted)
- Usage similar to cp
  - `scp [source] [destination]`
- Transferring to remote host
  - `scp /home/username/doc.txt username@ugrad.seas.ucla.edu:/home/user/docs`
  - Transferring from remote host
    - `scp username@ugrad.seas.ucla.edu:/home/user/docs/foo.txt /home/username`

Digital signature

- Protect **integrity** of the documents
  - Receiver received the document that the sender intended
- Digital signature is extra data attached to the document that can be used to check **tampering**
- Message digest
  - **Shorter** version of the document
  - Generated using **hashing** algorithms (SHA-2)
  - Even a slight change in the original document will change the message digest with **high probability**

Digital signature

Verifies document integrity, but does it prove origin? and who is the Certificate Authority?



Hints for Assignment 9

Cite Jin Wang

Lab9: Important notice

- Do not use virtual machines, please boot from USB or DVDs.
- Form groups of 2 or 3 people to finish the lab
- For the lab log, you can't just copy and paste the commands in this slides. Please explain the steps (what is the function of each command) in details to show that you understand the whole process.

Lab 9: Task

- Securely login to each others' computers
  - Use SSH (OpenSSH)
- Use key-based authentication
  - Generate key pairs
- Make logins convenient
  - type your passphrase once and be able to use SSH to connect to any other host without typing any passwords or passphrases
- Use port forwarding to run a command on a remote host that displays on your host

Lab Environment Setup

- Ubuntu
  - Make sure you have openssh-server and openssh-client installed
  - Check: *$ dpkg --get-selections | grep openssh*

    should output:
    - openssh-server install
    - openssh-client install
  - If not:
    - $ sudo apt-get update
    - *$ sudo apt-get install openssh-server*
    - *$ sudo apt-get install openssh-client*

Server Steps

- Generate public and private keys
  - $ *ssh-keygen* (by default saved to ~/.ssh/is_rsa and id_rsa.pub) – don't change the default location
- Create an account for the client on the server
  - *$ sudo useradd –d /home/<homedir_name> –m <username>*
  - *$ sudo passwd <username>*
- Create .ssh directory for new user
  - *$ cd /home/<homedir_name>*
  - *$ sudo mkdir .ssh*
- Change ownership and permission on .ssh directory
  - *$ sudo chown –R username .ssh*
  - *$ sudo chmod 700 .ssh*
- Optional: disable password-based authentication
  - *$ emcas /etc/ssh/sshd_config*
  - change PasswordAuthentication option to no

Client Steps

- Generate public and private keys
  - *$ ssh-keygen*
- Copy your public key to the server for key-based authentication (~/.ssh/authorized_keys)
  - *$ ssh-copy-id –i UserName@server_ip_addr*
- Add private key to authentication agent (ssh-agent)
  - *$ ssh-add*
- SSH to server
  - *$ ssh UserName@server_ip_addr*
  - *$ ssh –X UserName@server_ip_addr*
- Run a command on the remote host
  - *$ xterm, $ gedit, $ firefox*, etc.

How to Check IP Addresses

- *$ ifconfig*
  - configure or display the current network interface configuration information (IP address, etc.)
- *$ ping <ip_addr>(**p**acket **in**ternet **g**roper)*
  - Test the reachability of a host on an IP network
  - measure round-trip time for messages sent from a source to a destination computer
  - Example: $ ping 192.168.0.1, $ ping google.com

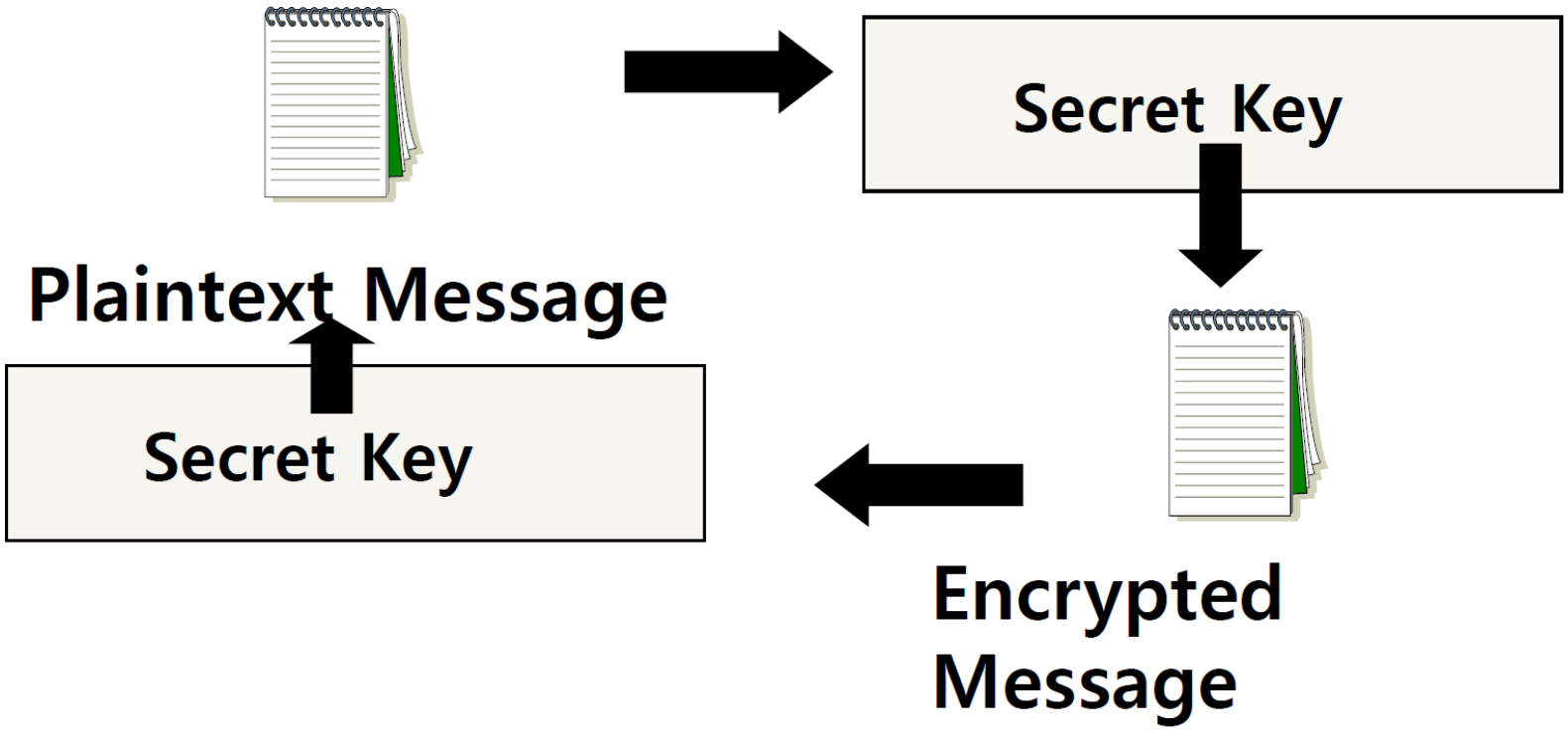
Homework 9

- Answer 2 questions in the file **`hw.txt`**
- Generate a key pair with the GNU Privacy Guard's commands
  - `$ gpg --gen-key` (choose default options)
- Export public key, in ASCII format, into **`hw-pubkey.asc`**
  - `$ gpg --armor --output hw-pubkey.asc --export 'Your Name'`
- Make a tarball of the above files + **`log.txt`** and zip it with gzip to produce `hw.tar.gz`
  - `$ tar -cf hw.tar <files>`
  - `$ gzip hw.tar` -> creates hw.tar.gz
- Use the private key you created to make a detached clear signature `hw.tar.gz.sig` for `hw.tar.gz`
  - `$ gpg --armor --output hw.tar.gz.sig --detach-sign hw.tar.gz`
- Use given commands to verify signature and file formatting
  - These can be found at the end of the assignment spec

CS 35L Software Construction Lab
Week 7 – Digital Signature

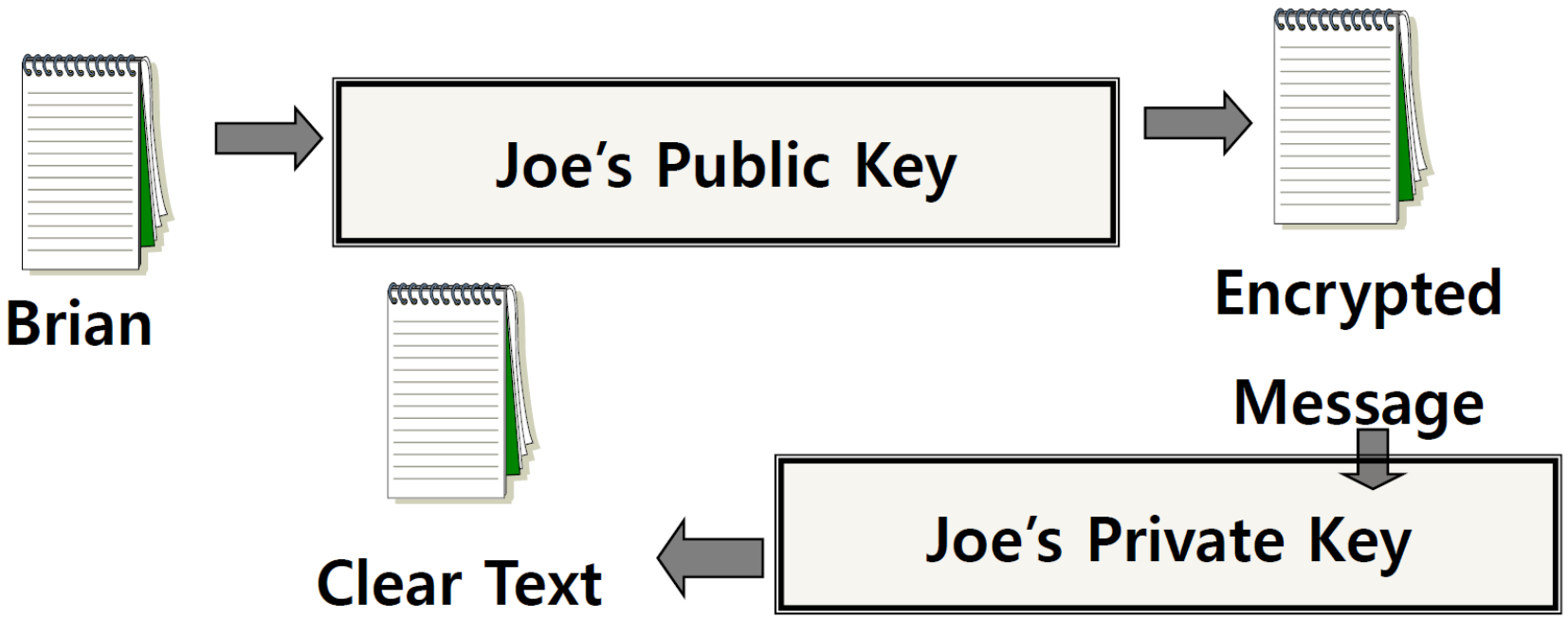
Secret Key (symmetric) Cryptography

- A single key is used to both encrypt and decrypt a message



Public Key (asymmetric) Cryptography

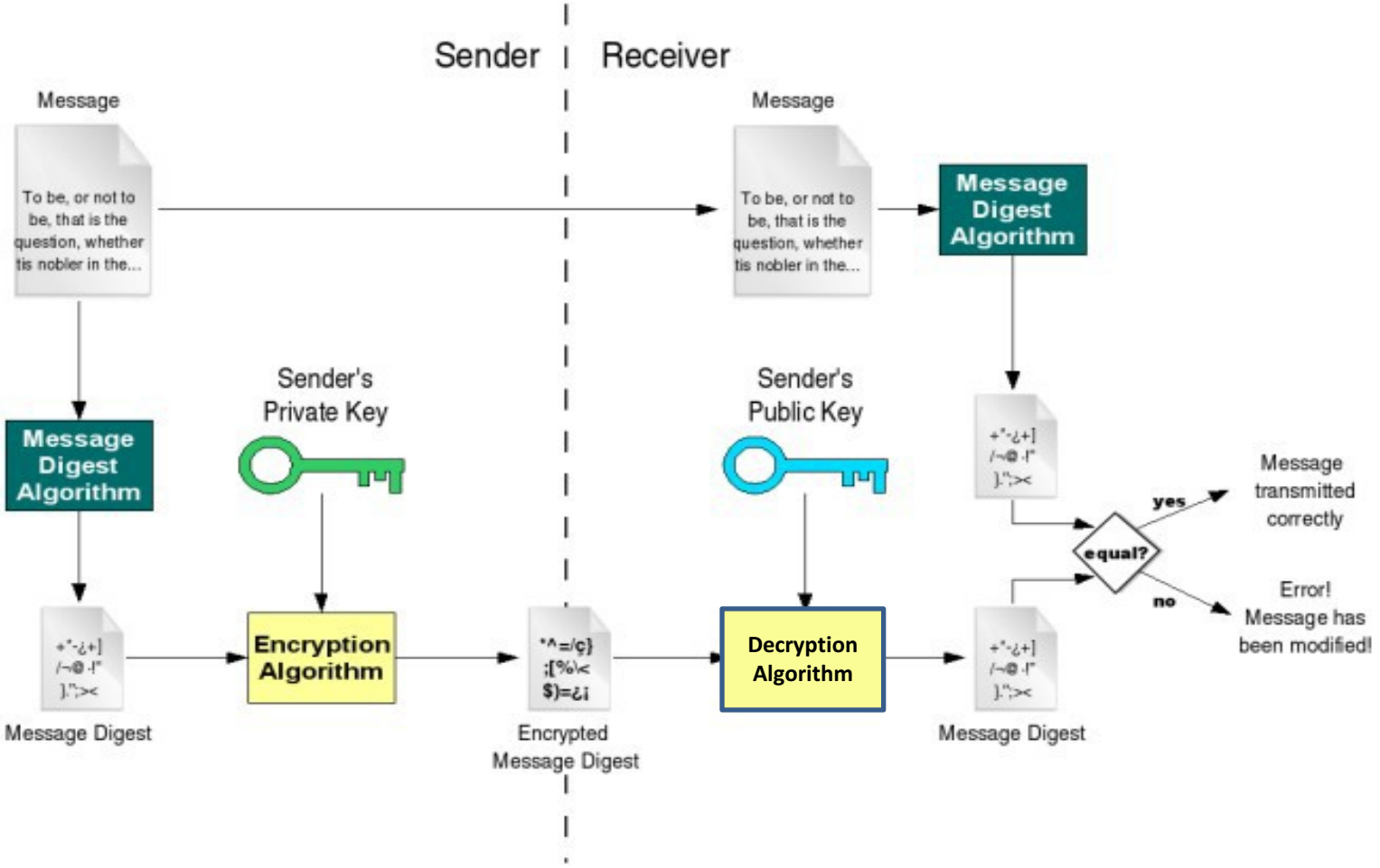
- Two keys are used: a public and a private key. If a message is encrypted with one key, it has to be decrypted with the other.



Digital Signature

- An electronic stamp or seal
  - almost exactly like a written signature, except more guarantees!
- Is appended to a document
  - Or sent separately (detached signature)
- Ensures data integrity
  - document was not changed during transmission

Digital Signature



Steps for Generating a Digital Signature

SENDER:

1) Generate a *Message Digest*
   - The message digest is generated using a set of hashing algorithms
   - A message digest is a 'summary' of the message we are going to transmit
   - Even the slightest change in the message produces a different digest
2) Create a Digital Signature
   - The message digest is encrypted using the sender's *private* key. The resulting encrypted message digest is the *digital signature*
3) Attach digital signature to message and send to receiver

Steps for Generating a Digital Signature

RECEIVER:

1) Recover the *Message Digest*
   - Decrypt the digital signature using the sender's public key to obtain the message digest generated by the sender
2) Generate the Message Digest
   - Use the same message digest algorithm used by the sender to generate a message digest of the received message
3) Compare digests (the one sent by the sender as a digital signature, and the one generated by the receiver)
   - If they are not *exactly the same* => the message has been tampered with by a third party
   - We can be sure that the digital signature was sent by the sender (and not by a malicious user) because *only* the sender's public key can decrypt the digital signature and that public key is proven to be the sender's through the certificate. If decrypting using the public key renders a faulty message digest, this means that either the message or the message digest are not exactly what the sender sent.

Detached Signature

- Digital signatures can either be *attached* to the message or *detached.*
- A detached signature is stored and transmitted separately from the message it signs.
- Commonly used to validate software distributed in compressed tar files.
- You can't sign such a file internally without altering its contents, so the signature is created in a separate file.

Homework 7

- Answer 2 questions in the file **`hw.txt`**
- Generate a key pair with the GNU Privacy Guard's commands
  - `$ gpg --gen-key` (choose default options)
- Export public key, in ASCII format, into **`hw-pubkey.asc`**
  - `$ gpg --armor --output hw-pubkey.asc --export 'Your Name'`
- Make a tarball of the above files + **`log.txt`** and zip it with gzip to produce `hw.tar.gz`
  - `$ tar -cf hw.tar <files>`
  - `$ gzip hw.tar` -> creates hw.tar.gz
- Use the private key you created to make a detached clear signature `hw.tar.gz.sig` for `hw.tar.gz`
  - `$ gpg --armor --output hw.tar.gz.sig --detach-sign hw.tar.gz`
- Use given commands to verify signature and file formatting
  - These can be found at the end of the assignment spec

Reference

- https://www.gnupg.org/gph/en/manual/x135.html

CS 35L Software Construction Lab
Week 8 – Dynamic Linking

Library

- A **library** is a package of code that is meant to be reused by many programs.
- A C++ library comes in two pieces
- Header file : defines the functionality the library is exposing (offering) to the programs using it
- Precompiled binary: contains the implementation of that functionality pre-compiled into machine language

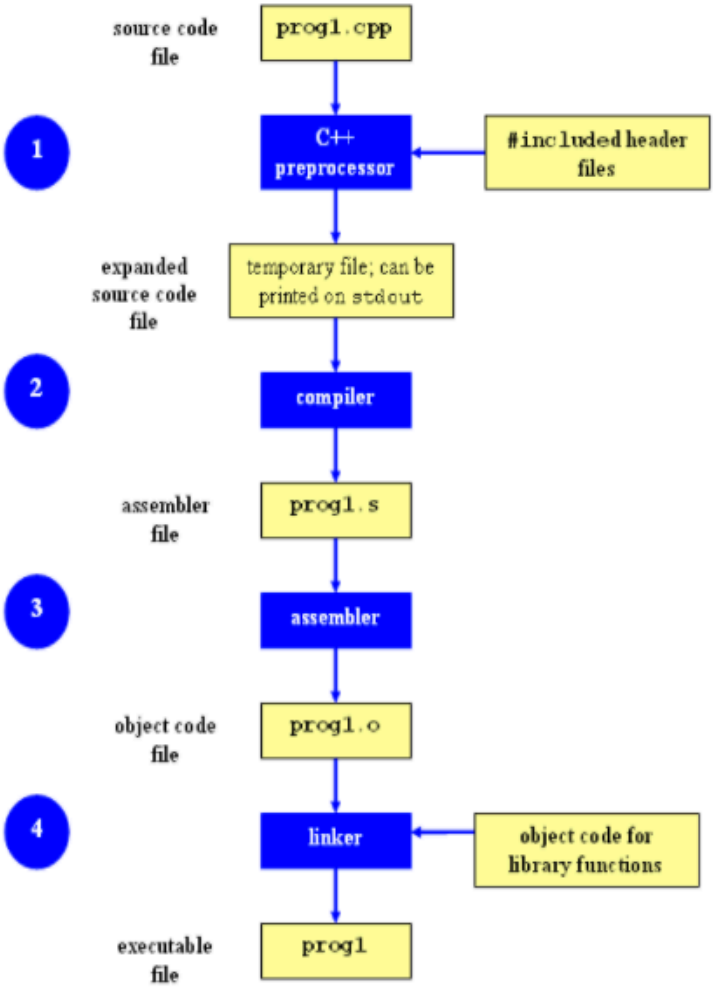
Question: Why do we want library code to be precompiled?

Library

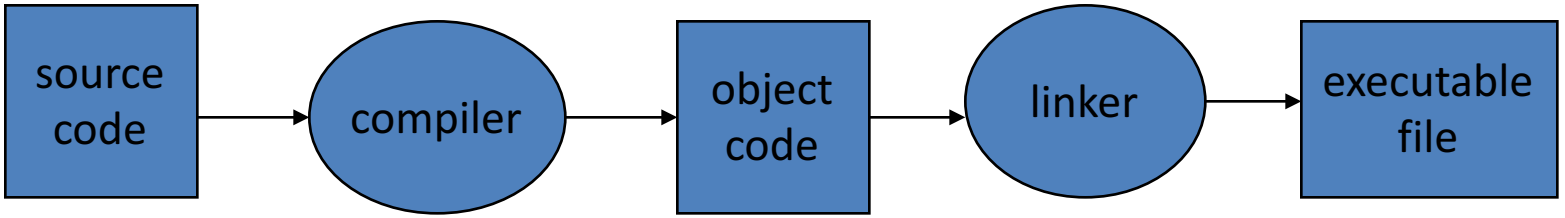
- Two types of libraries
  - static libraries
  - dynamic libraries
- Static Library (archive)
  - Windows uses .lib extension, linux uses .a (archive) extension.
  - Used in static linking
- Dynamic Library (shared object)
  - Windows uses .dll (dynamic link library) extension, Linux uses .so (shared object) extension
  - Used in dynamic linking and dynamic loading

Compilation Process

- Preprocessor
  - Expand Header includes, macros, etc
  - -E option in gcc to show the resulting code
- Compiler
  - Generates machine code for certain architecture
- Linker
  - Linkall modules together
  - Address resolution
- Loader
  - Loads the executable to memory to start execution



Building an executable file



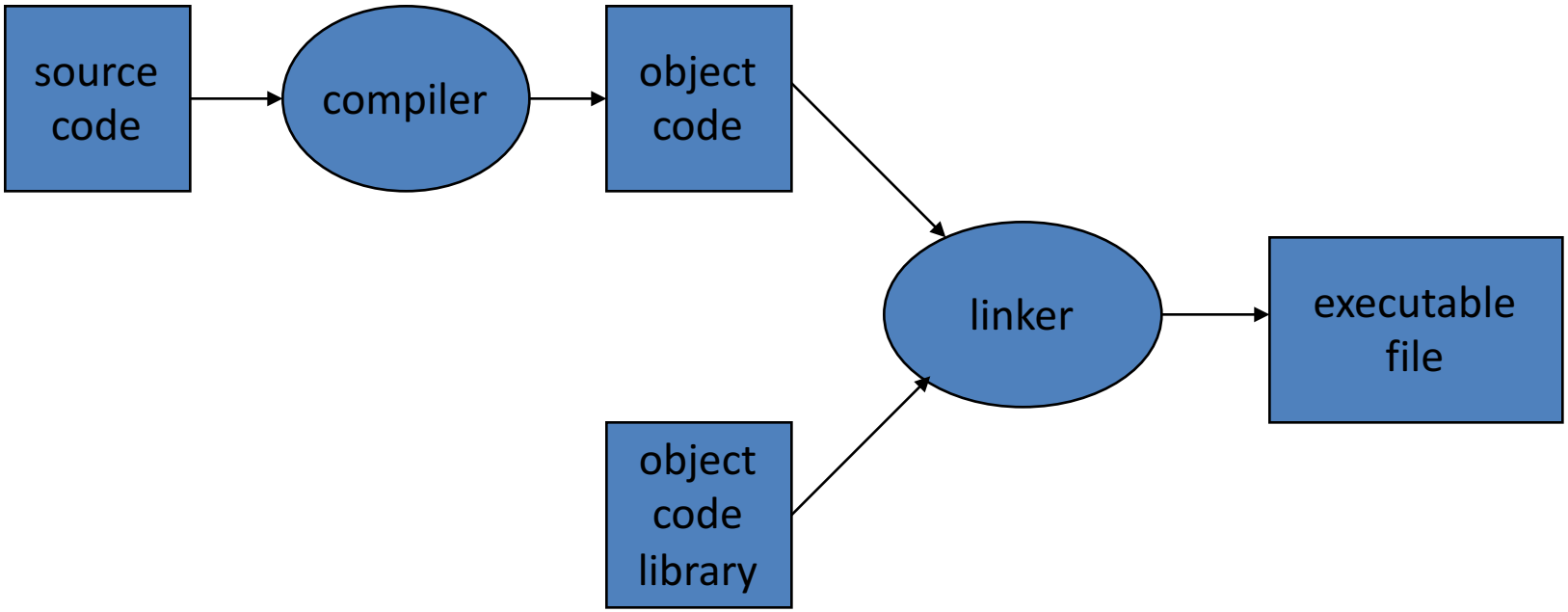
Translates programming language statements into cpu's machine-language instructions

Adjusts any memory references to fit the Operating Sytem's memory model

Static Linking

- Carried out only once to produce an executable file
- If static libraries are called, the linker will copy all the modules referenced by the program to the executable
- Static libraries are typically denoted by the .a file extension

Linking libraries

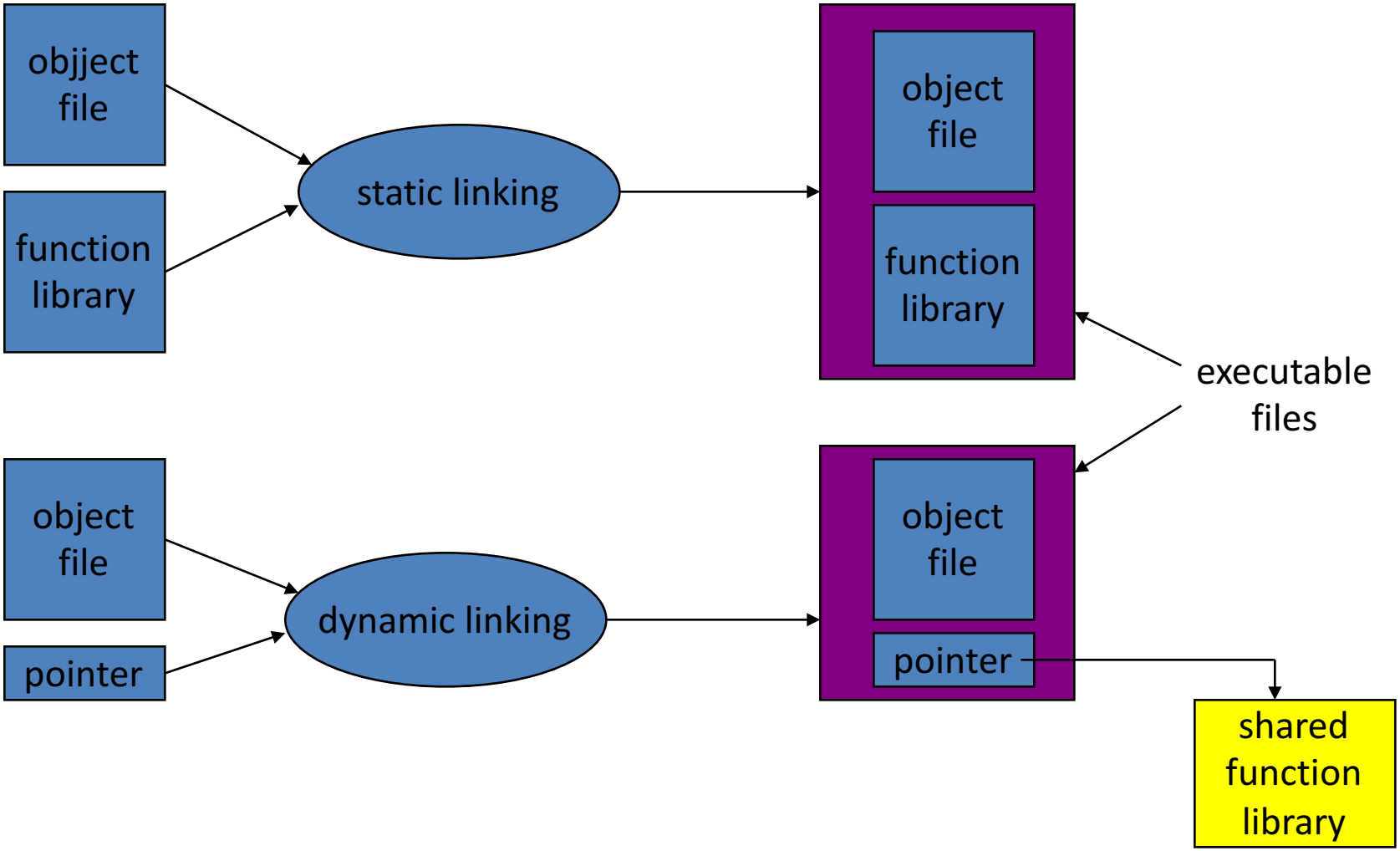


A previously compiled collection of standard program functions

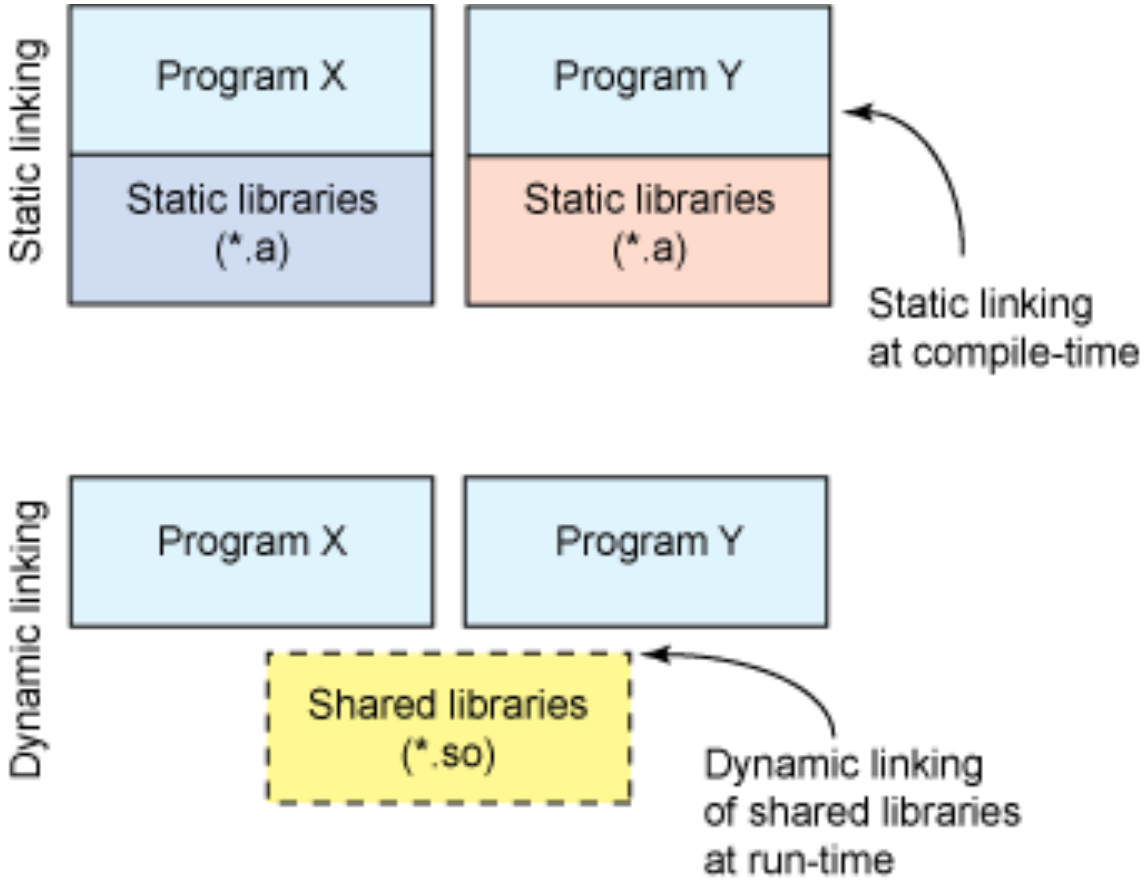
Dynamic Linking

- If shared libraries are called:
  - Only copy a little reference information when the executable file is created
  - Complete the linking during loading time or running time
- Dynamic libraries are typically denoted by the .so file extension
  - .dll on Windows

Smaller is more efficient



Q: What are the pros and cons?



Linking and Loading

- Why isn't everything written as just one **big** program, saving the necessity of linking?
  - Efficiency: if just one function is changed in a 100K line program, why recompile the whole program? Just recompile the one function and relink.
  - Multiple-language programs
  - Other reasons?

Dynamic linking

- Dynamic vs. static linking resulting size

```
$ gcc -static hello.c -o hello-static
$ gcc hello.c -o hello-dynamic
$ ls -l hello
     80 hello.c
  13724 hello-dynamic
    383 hello.s
1688756 hello-static
```

Advantages of dynamic linking

- The executable is typically smaller
- When the library is changed, the code that references it does not usually need to be recompiled
- The executable accesses the .so at run time; therefore, multiple programs can access the same .so at the same time
  - Memory footprint amortized across all programs using the same .so

Disadvantages of dynamic linking

- Performance hit
  - Need to load shared objects (at least once)
  - Need to resolve addresses (once or every time)
- What if the necessary dynamic library is missing?
- What if we have the library, but it is the wrong version?

Lab 8

- Write and build simple “cos(0.5)” program in C
  - Use `ldd` to investigate which dynamic libraries your hello world program loads
  - Use `strace` to investigate which system calls your hello world program makes
- Use “`ls /usr/bin | awk 'NR%101==SID%101'`” to find ~25 linux commands to use `ldd` on
  - Record output for each one in your log and investigate any errors you might see
  - From all dynamic libraries you find, create a sorted list
    - Remember to remove the duplicates!

CS 35L Software Construction Lab
Week 8 – Dynamic Linking

Anatomy of Linux shared libraries

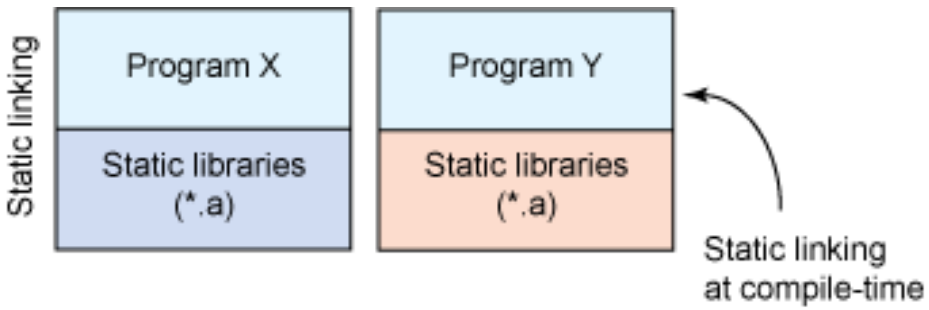
- Libraries - to package similar functionality → modular programming
- Linux supports two types

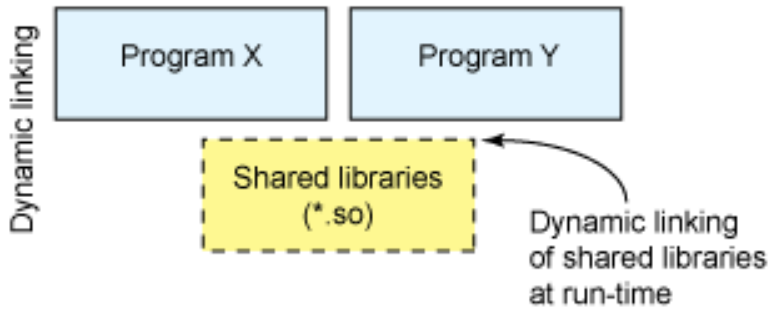
static library

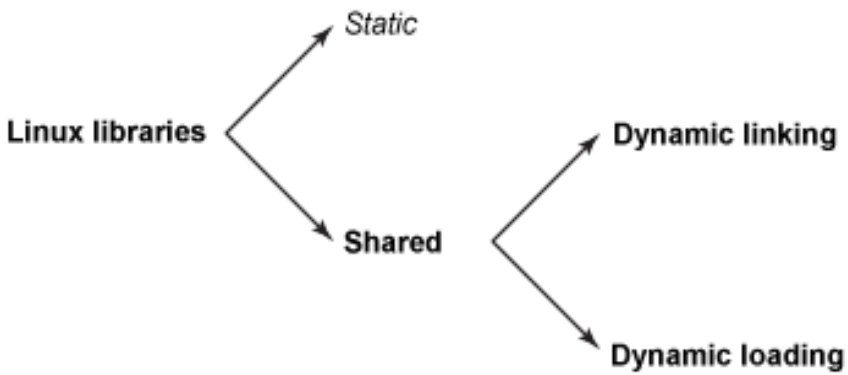
functionality to bind to a program statically at compile-time

dynamic library

functionality to bind to a program dynamically at run-time







dynamic linking - have Linux load the library upon execution

dynamic loading - selectively call functions with the library in a process

http://www.ibm.com/developerworks/library/l-dynamic-libraries/

Dynamic Loading

to let an application load and link libraries itself

- application can specify a particular library to load, then
- application can call functions within that library

load shared libraries from disk (file) into memory and re-adjust its location

done by a library named ld-linux.so.2

the Dynamic Loading API

dlopen - makes an object file accessible to a program

void *dlopen(const char *file, int mode);

RTLD NOW → relocate now; RTLD LAZY → to relocate when needed;

dlsym - gives resolved address to a symbol within this object

void *dlsym(void *restrict handle, const char *restrict name);

check char *dlerror(); if an error occurs

dlerror - returns a string error of the last error that occurred

dlclose - closes an object file

Dynamic loading

```
#include <stdio.h>
#include <dlfcn.h>

int main(int argc, char* argv[]) {
 int i = 10;
 void (*myfunc)(int *); void *dl_handle;
 char *error;
 dl_handle = dlopen("libmymath.so", RTLD_LAZY);//RTLD_NOW
 if(!dl_handle) {
  printf("dlopen() error - %s\n", dlerror()); return 1;
 }
 //Calling mul5(&i);
 myfunc = dlsym(dl_handle, "mul5"); error = dlerror();
 if(error != NULL) {
  printf("dlsym mul5 error - %s\n", error); return 1;
 }
 myfunc(&i);
  //Calling add1(&i);
  myfunc = dlsym(dl_handle, "add1"); error = dlerror();
  if(error != NULL) {
   printf("dlsym add1 error - %s\n", error); return 1;
  }
  myfunc(&i);
  printf("i = %d\n", i);
  dlclose(dl_handle);
  return 0;
}
```

Creating static and shared libs in GCC

- mymath.h

```
#ifndef _ MY_MATH_H

#define _ MY_MATH_H

void mul5(int
*i);
void add1(int
*i);
#endif
```

- mul5.c

```
#include
"mymath.h"
void mul5(int
*i)
{

  *i *= 5;


}
```

- add1.c

```
#include
"mymath.h"
void add1(int
*i)
{

  *i += 1;


}
```

- gcc -c mul5.c -o mul5.o
- gcc -c add1.c -o add1.o
- ar -cvq libmymath.**a** mul5.o add1.o ----> (static lib)
- gcc -**shared** -fpic -o libmymath.**so** mul5.o add1.o -----> (shared lib)

Attributes of Functions

- Used to declare certain things about functions called in your program
  - Help the compiler optimize calls and check code
- Also used to control memory placement, code generation options or call/return conventions within the function being annotated
- Introduced by the **attribute** keyword on a declaration, followed by an attribute specification inside double parentheses

Attributes of Functions

- `__attribute__ ((__constructor__))`
  - Is run when `dlopen()` is called
- `__attribute__ ((__destructor__))`
  - Is run when `dlclose()` is called
- Example:

```
__attribute__ ((__constructor__))
void to_run_before (void) {
    printf("pre_func\n");
}
```

Homework 8

the homework - to split an application into dynamically linked modules
randall.c = randcpuid.c + randlibhw.c + randlibsw.c + randmain.c

randall.c =
randcpuid.c + randlibhw.c + randlibsw.c + randmain.c

1. build the libraries
2. load the libraries
3. run the functions in libraries

Homework 8

Flags:

gcc -shared -fPIC greeting-fr.c -o greeting-fr.so

gcc -ldl -Wl,-rpath=. greeting-dl.c -o greet-dl

- -fPIC to output position independent code
- -lmylib to link with \libmylib.so"
- -L to nd .so les from this path, default is /usr/lib
- -Wl,rpath=dir to set rpath option to be dir to linker (by using -Wl)
- -shared to build a shared object

Attribute of functions:

__attribute__ ((constructor)) to run when dlopen() is called

__attribute__ ((destructor)) to run when dlclose() is called

Homework 8

- Divide randall.c into dynamically linked modules and a main program. We don’t want resulting executable to load code that it doesn't need (dynamic loading)
  - **randcpuid.c**: contains code that determines whether the current CPU has the RDRAND instruction. Should include randcpuid.h and include interface described by it.
  - **randlibhw.c**: contains the hardware implementation of the random number generator. Should include randlib.h and implement interface described by it.
  - **randlibsw.c**: contains the software implementation of the random number generator. Should include randlib.h and implement interface described by it.
  - **randmain.c**: contains the main program that glues together everything else. Should include randcpuid.h but not randlib.h. Depending on whether the hardware supports the RDRAND instruction, this main program should dynamically load the hardware oriented or software oriented implementation of randlib.

Homework 8

- Stitch the files together via static and dynamic linking to create the program
- randmain.c must use *dynamic loading, dynamic linking* to link up with randlibhw.c and randlibsw.c (using randlib.h)
- Write the randmain.mk makefile to do the linking

Homework 8

- randall.c outputs N random bytes of data

  Look at the code and understand it
  - Helper functions that check if hardware random number generator is available, and if it is, generates number
    - Hw RNG exists if RDRAND instruction exists
    - Uses cpuid to check whether CPU supports RDRAND (30th bit of ECX register is set)
  - Helper functions to generate random numbers using software implementation (/dev/urandom)
  - Main function
    - Checks number of arguments (name of program, N)
    - Converts N to long integer, prints error message otherwise
    - Uses helper functions to generate random number using hw/sw