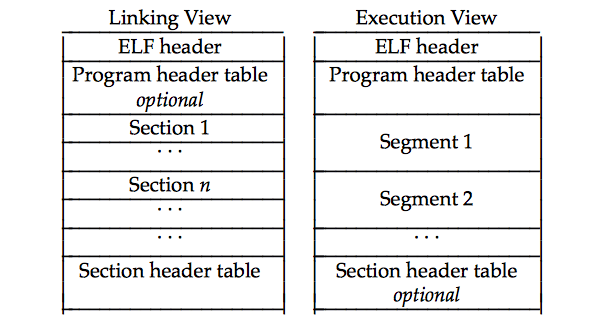
Test

The offset tells you where in the block

The index tells you what position

The tag is for look up.

When look for, check if the index is good, then check the tag. If the tag is different then it’s a miss.



The valid bit indicates whether there is a physical page that exist for the requested virtual page

16 bit virtual address

page size =8KB

how many page table entries do we have: 2^16/2^13=8

figure out physical address based on virtual address:

1. write in binary

2. find virtual page number and page offset

3. find physical page number

4. concatenate and figure out the physical address.

Page fault: like a cache miss.

LRU, evict blocks

Dynamic Memory Allocators: Chapter 9.9

>>Writing a heap simulator

**Building a heap with explicit free list**

Void \*ptr=malloc(ptr)

Free list:

A linked list for all the free blocks in the heap

1. mark the block as allocated

2. remove it from the linked list

if too big, we split the block and then do it.

\*\*\*split the block, allocate a new header and a new footer.

If amount+32=payload

We don’t split.

Originally there’s a huge block, with header and footer.

When we allocate, we split one and the other one(amount+32) 🡪32 accounts for the header and footer.

P’=P-(amount+32)

Hdr\_prev(v)

Hdr\_prev(hdr\_nxt)

Free:

Free the one before:

Anchor points to whatever is being freed most recently

If it’s next to it, we coasclets (merge)

Zero out header and footer after using them

Free the one after:

1. remove A from FL (free list)

2. add B to FL

3. extend B into A

Free the one in the middle:

1. check pre~A if free, extend A

2. Else add B to freeliu

3. if next free rm C and extend

Write back cache:

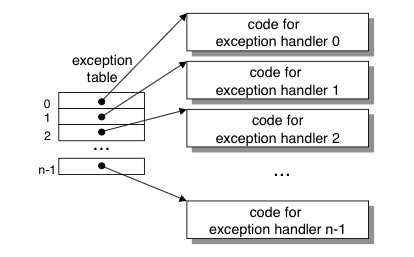
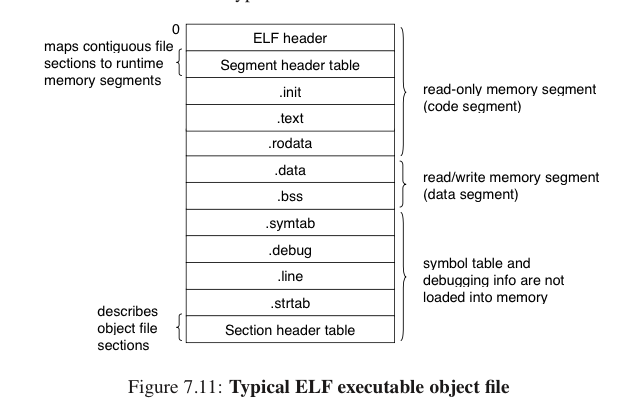
A [caching](http://www.webopedia.com/TERM/C/cache.html) method in which modifications to data in the cache aren't copied to the cache source until absolutely necessary.

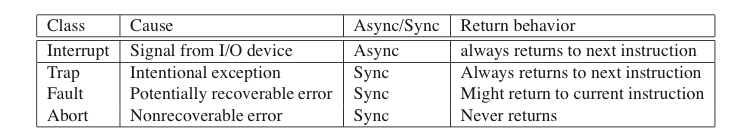
Write through cache:

**Write-through cache** directs write I/O onto cache and through to underlying permanent storage before confirming I/O completion to the host.

**Write-around cache** is a similar technique to write-through cache, but write I/O is written directly to permanent storage, bypassing the cache.

* 􏰔  *Miss rate.* The fraction of memory references during the execution of a program, or a part of a  program, that miss. It is computed as *# misses*􏱗*# references*.
* 􏰔  *Hit rate.* The fraction of memory references that hit. It is computed as 􏰞 􏰌 *miss rate*.
* 􏰔  *Hit time.* The time to deliver a word in the cache to the CPU, including the time for set selection, line identification, and word selection. Hit time is typically 1 to 2 clock cycle for L1 caches.
* 􏰔  *Miss penalty.* Any additional time required because of a miss. The penalty for L1 misses served from L2 is typically 5 to 10 cycles. The penalty for L1 misses served from main memory is typically 25 to 100 cycles.





The getppid function returns the PID of its *parent* (i.e., the process that created the calling process).

The fork function is interesting (and often confusing) because it is called *once* but it returns *twice*: once in the calling process (the parent), and once in the newly created child process.

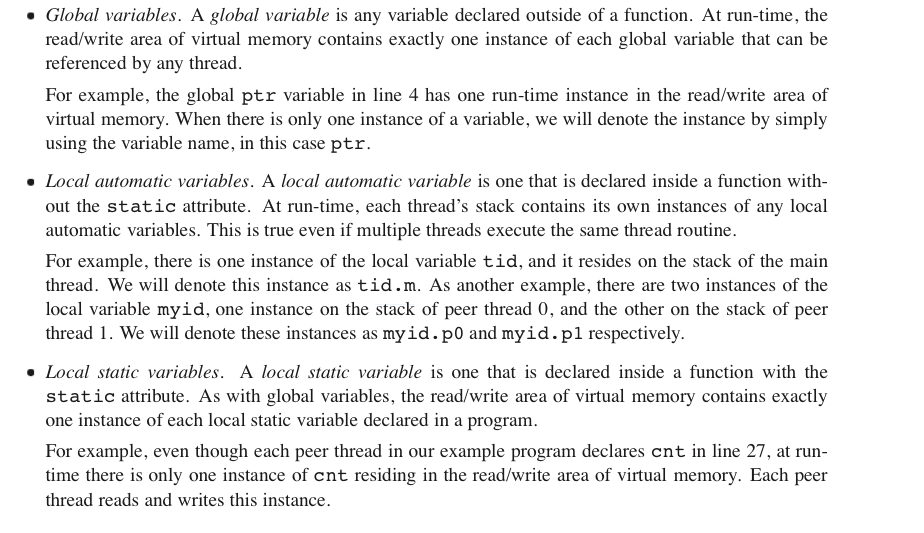
When a process terminates for any reason, the kernel does not remove it from the system immediately. Instead, the process is kept around in a terminated state until it is *reaped* by its parent. When the parent reaps the terminated child, the kernel passes the child’s exit status to the parent, and then discards the terminated process, at which point it ceases to exist. A terminated process that has not yet been reaped is called a *zombie*.

Thread execution differs from processes in some important ways. Because a thread context is much smaller than a process context, a thread context switch is faster than a process context switch. Another difference is that threads, unlike processes, are not organized in a rigid parent-child hierarchy.

When the main thread detects that the peer thread has terminated, it terminates itself (and the entire process) by calling exit.

The pthread join function blocks until thread tid terminates, assigns the (void \*) pointer returned by the thread routine to the location pointed to by thread return, and then *reaps* any memory resources held by the terminated thread.

At any point in time, a thread is *joinable* or *detached*. A joinable thread can be reaped and killed by other threads. Its memory resources (such as the stack) are not freed until it is reaped by another thread. In contrast, a detached thread cannot be reaped or killed by other threads. Its memory resources are freed automatically by the system when it terminates.



unsafe functions :

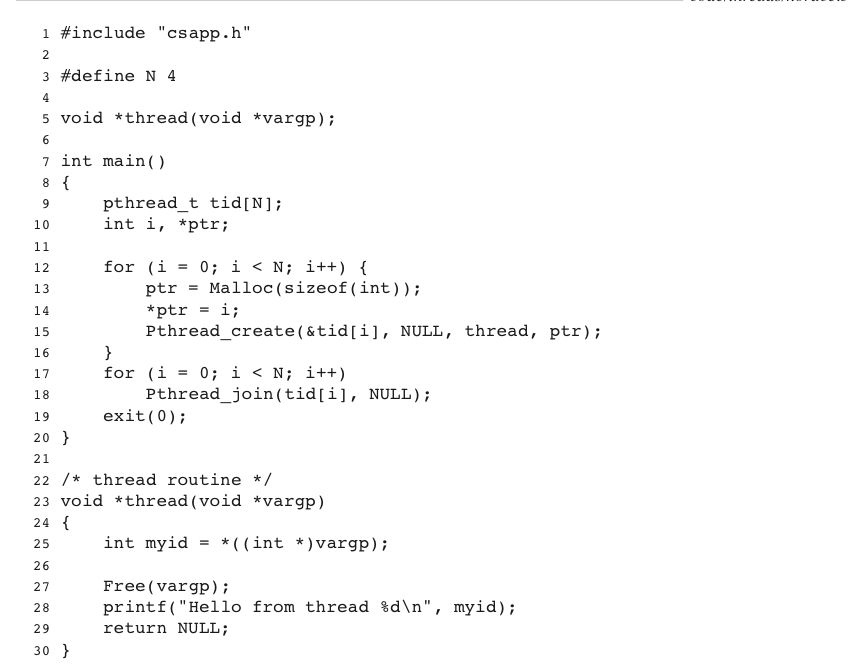
*Failing to protect shared variables.*

*Relying on state across multiple function invocations.*

*Returning a pointer to a static variable.*

*Calling thread-unsafe functions.*

A *race* occurs when the correctness of a program depends on one thread reaching point 􏰢 in its control flow before another thread reaches point 􏰣 .



Semaphores:

Restrict access.

Two operations:

Down: return (s>0)?—s:RETRY P

Up: ++s; V

Atomic: indivisible

S==1 unlocked

S==0 locked

P attempt to acquire lock s

V unlock s

Condition for writing: no one is reading and the write lock is available.

Exponent=1-bias

8 GB->33 bits

4kb -> 12 bits

VPN(virtual page number)=33-12=21.

How many entries: P=2^21= page table entries.

0X13477fad7

ad7: last 12 bits🡪page offset

virtual page number: 0x13477f

TLB is like a ache for page table

TLB example.

VA=20bits

PA=18 bits

Page size=4KB

TLB index=3 (there are 8 sets)

Physical page offset is the same as virtual page offset

**The return address is ebp+4**

If little endian

From highest memory to lowest memory

The first parameter is stored at ebp+8

The second parameter is stored at ebp+12;