EngSci Year 3 Winter 2022 Notes

Brian Chen

Division of Engineering Science
University of Toronto
https://chenbrian.ca

brianchen.chen@mail.utoronto.ca

Contents

1 CSC473: Advanced Algorithms **1**
1.1 Global Min-Cut 1

2 ECE568 Computer Security **2**
2.1 Refresher & Introduction 2
2.1.1 Security Fundamentals 3

3 ECE353 Operating Systems **3**
3.1 Kernel Mode 3
3.1.1 ISAs and Permissions 3
3.1.2 ELF (Executable and Linkable Format) 4
3.1.3 Kernel 6

Section 1

CSC473: Advanced Algorithms

Subsection 1.1

Global Min-Cut

Given an undirected, unweighted, and connected graph $G = (V, E)$, **return** the smallest set of edges that disconnects G

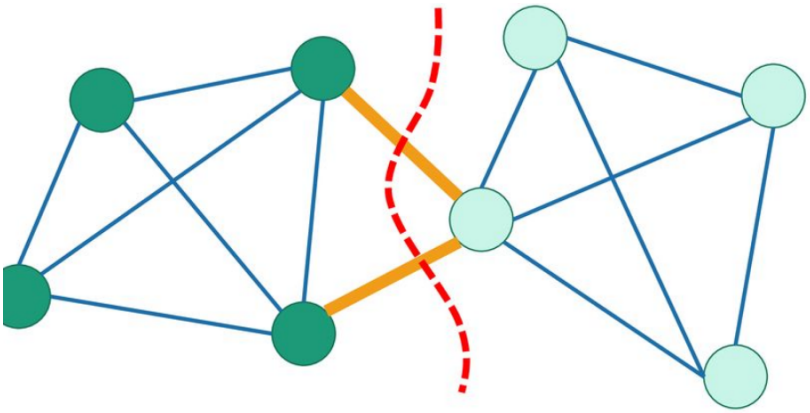

Figure 1. Example of global min-cut. Note that the global min-cut is not necessarily unique

Lemma 1 | If the min cut is of size $\geq k$, then G is k-edge-connected

An example of where this may be useful is in computer networks where we can measure the resiliency of a network by how many cuts must be made before a vertex (or many) get disconnected

It may be more convenient to return a set of vertices instead

Definition 1

$$S, T \subseteq V, S \cap T = \varnothing \tag{1.1}$$

$$E(S, T) = \{(u, v) \in E : u \in S, v \in T\} \tag{1.2}$$

The global min-cut is to output $S \subseteq V$ such that $S \neq \varnothing, S \neq V$, such that $E(S, V \setminus S)$ is minimized.

Comment

Note that the min-cut-max-flow problem is somewhat of a dual to the global min-cut problem; the min-cut-max-flow problem imposes a few more constraints than the global min-cut algorithm i.e. having a directed and weighted graph as well as the notion of a source or sink.

- **Input:** Directed, weighted, and connected $G = (V, E)$, $s \in V, t \in V$
- **Output** : S such that $s \in S, t \notin S$ such that $|E(S, V \setminus S)|$ is minimized

We can kind of intuitively see that the global min-cut can be taken to the minimum of all max-flows across the graph. So we can take the max-flow solution and then reduce it to find the global min cut.

Question: how many times will we have to run max-flow to solve the global min-cut problem? Naively, we may fix t to be an arbitrary node, then try every other $s \neq t$ to find the $s - t$ min-cut to get the best global min-cut.

We know from previous courses that the Edmonds-Karp max-flow algorithm will run in $O(nm^2) = O(n^5)$, which makes our global min-cut algorithm $O(n^6)$. However, there is a paper recently published which gives an algorithm for min-cut in nearly linear time, i.e $O(m^{1-O(1)}) = O(n^2)$ which gives a global min-cut runtime of $O(n^3)$.

A randomized algorithm will be presented that solves this problem in $O(n^2 \log^2 n)$

Definition 2 The **Contraction** operation takes an edge $e = (u, v)$ and *contracts* it into a new node w such that all edges connected to u, v now connect to w and u, v are removed. Note that the contracted nodes can be supernodes themselves.

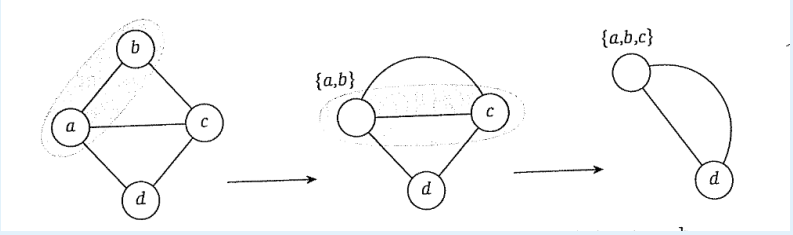

Figure 2. Example of a series of contractions

CONTRACTION$(G = (V, E))$

1 **while** G has more than 2 supernodes
2 Pick an edge $e = (u, v)$ uniformly at random
3 Contract e, remove self-loops
4 Output the cut $(S, V \setminus S)$ corresponding to the two super nodes

The contraction algorithm then recurses on G', choosing an edge uniformly at random and then contracting it. The algorithm terminates when it reaches *a* G' with only two supernodes v_1, v_2. The sets of nodes contracted to form each supernode $S(v_1), S(v_2)$ form a partition of V and are the cut found by the algorithm.

1.1.1 Analysis

The algorithm is still random, so there's a chance that it won't find the real global min-cut. With some analysis we will show that the success polynomial is not exponential as one may think, but really only polynomially small. Therefore by running the algorithm a polynomial number of times and returning the best cut identified we can find a global min-cut with high probability.

Lemma 2 The contraction algorithm returns a global min cut with probability at least $\frac{1}{\binom{n}{2}}$

Proof

Take a global min-cut (A, B) of G and suppose it has size k, i.e. there is a set F of k edges with one end in A and the other in B. If an edge in F gets contracted then a node of A and a node in B would get contracted together and then the algorithm would no longer output (A, B), a global min-cut.
An upper bound on the probability that an edge in F is contracted is the ratio of k to the size of E. A lower bound on the size of E can be imposed by noting that if any node v has degree $< k$ then $(v, V \setminus v)$ would form a cut of size less than k – which contradicts our first assumption that (A, B) is a global min-cut. So the probability than an edge in F is contracted at any step is

$$\frac{k}{(\frac{k}{2})} = \frac{2}{n} \tag{1.3}$$

Let's inspect the algorithm after j iterations. There will be $n - j$ supernodes in G' and we can take that no edge in F has been contracted yet. Every cut of G' is a cut of G, so there are at least k edges incident to every supernode of G'[1]. Therefore G' has at least $\frac{1}{2}k(n-j)$ edges, and so the probability than an edge of F is contracted in $j+1$ is at most

$$\frac{k}{\frac{1}{2}k(n-j)} = \frac{2}{n-j} \tag{1.4}$$

The global min-cut will be actually returned by the algorithm if no edge of F is contracted in iterations $1 - n$. The probability of

□

Comment

Note that there are two types of randomized algorithms:

- Monte carlo algorithms: bound on worst-case time & produces a correct answer with a probability $\geq$ some constant
- Las vegas algorithms: bound on the expected value of running time, but the output is always correct

Our contraction algorithm is a monte-carlo algorithm

Section 2

ECE568 Computer Security

Subsection 2.1

Refresher & Introduction

Software systems are ubiquitous and critical. Therefore it is important to learn how to protect against malicious actors. This course covers attack vectors and ways to design software securely

Data representation: It's important to recognize that data is just a collection of bits and it is up to us to tell the computer how it should be interpreted. Oftentimes we can make assumptions, for example assume that an int is an int. But what if we end up being wrong about it? Many security exploits rely on data being interpreted in a different way than originally intended. For example,

```c
unsigned long int h = 0x6f6c6c6548; // ascii for hello
unsigned long int w = 431316168567; // ascii for world
printf("%s %s", (char*) h, (char*) w);
```

Listing 1: An innocent example of where we should be careful about data representation. This prints hello world

This courses makes use of Intel assembler. TLDR:

- 6 General-purpose registers
- RAX (64b), EAX(32b), AX(16b), AH/AL(8b), etc

Note that the stack grows downwards and the heap grows upwards. Stack overflows can occur and can be a source of vulnerability.

GDB offers some tools for examining stacks

- `break`: create a new breakpoint
- `run`: start a new process
- `where`: list of current stack frames
- `up/down`: move between frames
- `info frame` display info on current frame
- `info args`: list function arguments
- `info locals`: list local variables
- `print`: display a variable
- `x` display contents of memory
- `fork`: Creates a new child process by duplicating the parent. The child has its own new unique process ID
- `exec`: Replaces the current process with a new process

The fork-exec technique is just a pair of `fork` and `exec` system calls to spawn a new program in a new process

2.1.1 Security Fundamentals

The three key components of security are:

- Confidentiality: the protection of data/resources from exposure, whether it be the content or the knowledge that the resource exists in the first place. Usually via organizational controls (security training), access rules, and cryptography.
- Integrity: Trustworthiness of data (contents, origin). Via monitoring, auditing, and cryptography.
- Availability: Ability to access/use a resource as desired. Can be hard to ensure; uptime, etc...

Together they form an acryonym: CIA. A system is considered secure if it has all three of these properties for a given time. The strength of cryptographic systems can be evaluated by the number of bits of entropy or their complexity. For example, a 128-bit key has $2^128 possible values. This would take a lot of time to break, and a 256-bit key even longer. Availability is harder to measure quantitatively and is instead traditionally measured qualitatively. For exampl$

Section 3

ECE353 Operating Systems

Subsection 3.1

Kernel Mode

3.1.1 ISAs and Permissions

There are a number of ISAs in use today; x86 (amd64), aarch64 (arm64), and risc-v are common ones. For purposes of this course we will study largely arm systems but will touch on the other two as well.

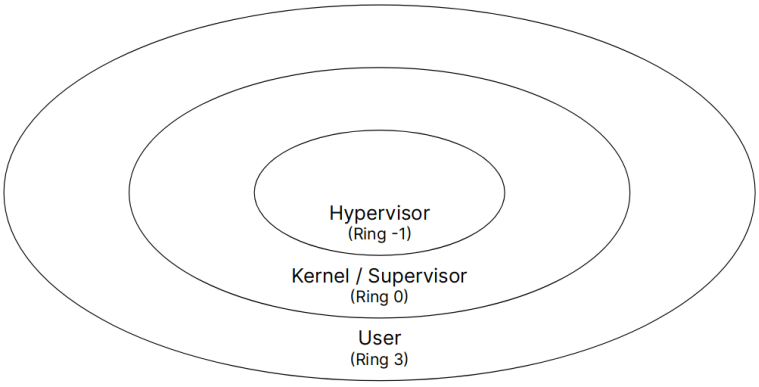

Figure 3. x86 Instruction access rings. Each ring can access instructions in its outer rings.

The kernel runs in, well, Kernel mode. **System calls** offer an interface between user and kernel mode[2].

The system call ABI for x86 is as follows:

Enter the kernel with a svc instruction, using registers for arguments:

- x8 — System call number
- x0 — 1st argument
- x1 — 2nd argument
- x2 — 3rd argument
- x3 — 4th argument
- x4 — 5th argument
- x5 — 6th argument

This ABI has some limitations; i.e. all arguments must be a register in size and so forth, which we generally circumvent by using pointers.

For example, the `write` syscall can look like:

```
1 ssize_t write(int fd, const void* buf, size_t count);
2 // writes bytes to a file description
```

[2] *Linux ha*

Note: API (ap programming i ABI (Application Interface). API communication inter two ints), ABI is how data, i.e. calling conv

3.1.2 ELF (Executable and Linkable Format)

- Aways starts with 4 bytes: `0x7F, 'E', 'L', 'F'`
- Followed byte for 32 or 64 bit architecture
- Followed by 1 byte for endianness

`readelf` can be used to read `ELF` file headers.
For example, `readelf -a $(which cat)` produces (output truncated)

Most file formats h
ferent starting signa
magic numbers

```
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              DYN (Position-Independent
  ↪  Executable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x32e0
  Start of program headers:          64 (bytes into file)
  Start of section headers:          33152 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         13
  Size of section headers:           64 (bytes)
  Number of section headers:         26
  Section header string table index: 25
```

`strace` can be used to trace systemcalls. For example let's look at the 168-byte hello-world example

This output is follow
formation about the
and section headers

```
0x7F 0x45 0x4C 0x46 0x02 0x01 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00
↪  0x00 0x00
0x02 0x00 0xB7 0x00 0x01 0x00 0x00 0x00 0x78 0x00 0x01 0x00 0x00 0x00
↪  0x00 0x00
0x40 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
↪  0x00 0x00
0x00 0x00 0x00 0x00 0x40 0x00 0x38 0x00 0x01 0x00 0x40 0x00 0x00 0x00
↪  0x00 0x00
0x01 0x00 0x00 0x00 0x05 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
↪  0x00 0x00
0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00
↪  0x00 0x00
0xA8 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0xA8 0x00 0x00 0x00 0x00 0x00
↪  0x00 0x00
0x00 0x10 0x00 0x00 0x00 0x00 0x00 0x00 0x08 0x08 0x80 0xD2 0x20 0x00
↪  0x80 0xD2
0x81 0x13 0x80 0xD2 0x21 0x00 0xA0 0xF2 0x82 0x01 0x80 0xD2 0x01 0x00
↪  0x00 0xD4
0xC8 0x0B 0x80 0xD2 0x00 0x00 0x80 0xD2 0x01 0x00 0x00 0xD4 0x48 0x65
↪  0x6C 0x6C
0x6F 0x20 0x77 0x6F 0x72 0x6C 0x64 0x0A
```

Listing 2: Note: This is for arm cpus

If we run this then we see that the program makes a `write` syscall as well as a `exit_group`

```
execve (" ./ hello_world " , [ " ./ hello_world " ] , 0 x7ffd0489de40
↪  /* 46 vars */ ) = 0
write (1 , " Hello world \ n " , 12) = 12
exit_group (0) = ?
+++ exited with 0 +++
```

Note that these strings are not null-terminated (null-termination is just a `c` thing) because we don't want to be unable to write strings with the null character to it.

```
execve("./hello_world_c", ["./hello_world_c"], 0x7ffcb3444f60 /* 46 vars */) = 0
brk(NULL)                               = 0x5636ab9ea000
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=149337, ...}) = 0
mmap(NULL, 149337, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f4d43846000
close(3)                                = 0
openat(AT_FDCWD, "/usr/lib/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0000C"..., 832) = 832
lseek(3, 792, SEEK_SET)                 = 792
read(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\201\336\t\36\251c\324"..., 68) = 68
fstat(3, {st_mode=S_IFREG|0755, st_size=2136840, ...}) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0)
  = 0x7f4d43844000
lseek(3, 792, SEEK_SET)                 = 792
read(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\201\336\t\36\251c\324"..., 68) = 68
lseek(3, 864, SEEK_SET)                 = 864
read(3, "\4\0\0\0\20\0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0", 32) = 32
```

Figure 4. A c hello world would load the stdlib before printing...

3.1.3 Kernel

The kernel can be thought of as a long-running program with a ton of library code which executes on-demand. Monolithic kernels run all OS services in kernel mode, but micro kernels run the minimum amount of servers in kernel mode. Syscalls are slow so it can be useful to put things in the kernel space to make it faster. But there are security reasons against putting everything in kernel mode.

3.1.4 Processes Syscalls

A process is like a combination of all the virtual resources; a "virtual GPU" (if applicable), memory (addr space), I/O, etc. The unique part of a struct is the PCB (Process Control Block) which contains all of the execution information. In Linux this is the `task`$_{s}truct which contains information about the process state, CPU registers, scheduling information, and so forth.$

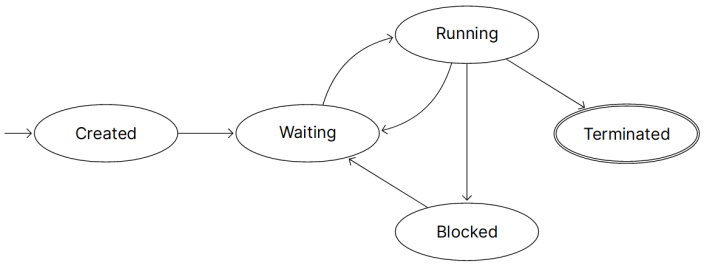

Figure 5. A possible process state diagram

These state changes are managed by the Process and OS[3] so that the OS scheduler can do its job. An example of where some of these states can be useful would be to free up CPU time while a process is in the Blocked state while waiting for IO. Process can either manage themselves (cooperative multitasking) or have the OS manage it (true multitasking). Most systems use a combination of the two, but it's important to note that cooperative multitasking is not true multitasking.

Context switching (saving state when switching between processes) is expensive. Generally we try to minimize the amount of state that has to be saved (the bare minimum is the registers). The scheduler decides when to switch. Linux currently uses the CFS[4].

In c most system calls are wrapped to give additional features and to put them more concretely in the userspace.

Process state c
be read in /pro
linux systems.

```c
#include <stdio.h>
#include <stdlib.h>

void fini(void) {
  puts("Do fini");
}

int main(int argc, char **argv) {
  atexit(fini);
  puts("Do main");
  return 0;
}
```

Figure 6. Demonstration of c feature to register functions to call on program exit

```c
int main ( int argc , char * argv []) {
  printf ("I 'm going to become another process \n" );
  char * exec_argv [] = {" ls " , NULL };
  char * exec_envp [] = { NULL };
  int exec_return = execve ("/ usr / bin / ls " , exec_argv ,
  ↪  exec_envp );
  if ( exec_return == -1) {
    exec_return = errno ;
    perror (" execve failed " );
    return exec_return ;
  }
  printf (" If execve worked , this will never print \n" );
  return 0;
}
```

Listing 3: Demo of execve turning current program to ls (executes program, wrapper around exec syscall)

```c
#include <sys/syscall.h>
#include <unistd.h>

int main (){
  syscall(SYS_exit_group, 0);
}

```

Listing 4: An example of using a raw syscall system exit instead of c's exit()