# Code Sandboxing

아아~.
제6화
예스맨이 인기를 끈다

온라인저지
만들고 싶다~.
......
큰 소리로
말하지 마,
등신아!

# 온라인 저지가 만들고 싶다

근데 어떻게 만들지?

# 온라인 저지가 동작하는 간략한 흐름

- 유저가 문제를 해결하는 프로그램의 코드를 서버에 제출한다.
- 서버는 유저가 제출한 코드를 실행/컴파일한다.
- 코드를 실행/컴파일하며, 시간/메모리 제한을 넘어서지 않는지 확인한다.
- 코드가 정답인지 확인한다.
- 결과를 유저에게 알려준다.

# 온라인 저지가 동작하는 간략한 흐름

- 유저가 문제를 해결하는 프로그램의 코드를 서버에 제출한다.
- 서버는 유저가 제출한 코드를 실행/컴파일한다.
- 코드를 실행/컴파일하며, 시간/메모리 제한을 넘어서지 않는지 확인한다.
- 코드가 정답인지 확인한다.
- 결과를 유저에게 알려준다.

# 온라인 저지가 동작하는 간략한 흐름

- 유저가 문제를 해결하는 프로그램의 코드를 서버에 제출한다.
- 서버는 유저가 제출한 코드를 실행/컴파일한다.
- 코드를 실행/컴파일하며, 시간/메모리 제한을 넘어서지 않는지 확인한다.
- 코드가 정답인지 확인한다.
- 결과를 유저에게 알려준다.

# 코드의 실행과 컴파일

채점 환경에서 코드의 실행과 컴파일은 아래와 같이 할 수 있다.

```rust
use nix::unistd::execv;

fn convert_args(args: Vec<&str>) -> Vec<CString> {
    args.into_iter().map(|arg| CString::new(arg).unwrap()).collect()
}

// compile
let binary = "/usr/bin/gcc";
let output_file = "output";
let args = convert_args(vec![binary, "input.c", "-O2", "-o", output_file]);
execv(&CString::new(binary).unwrap(), &args);

// execute
let output_file = CString::new(output_file).unwrap();
execv(&CString::new(output_file).unwrap(), &[output_file]);
```

# 코드의 실행과 컴파일

채점 환경에서 코드의 실행과 컴파일은 아래와 같이 할 수 있다.

```rust
use nix::unistd::execv;

fn convert_args(args: Vec<&str>) -> Vec<CString> {
    args.into_iter().map(|arg| CString::new(arg).unwrap()).collect()
}

// compile
let binary = "/usr/bin/gcc";
let output_file = "output";
let args = convert_args(vec![binary, "input.c", "-O2", "-o", output_file]);
execv(&CString::new(binary).unwrap(), &args);

// execute
let output_file = CString::new(output_file).unwrap();
execv(&CString::new(output_file).unwrap(), &[output_file]);
```

# 코드의 실행과 컴파일

채점 환경에서 코드의 실행과 컴파일은 아래와 같이 할 수 있다.

```
1   use nix::unistd::execv;
2
3   fn convert_args(args: Vec<&str>) -> Vec<CString> {
4       args.into_iter().map(|arg| CString::new(arg).unwrap()).collect()
5   }
6
7   // compile
8   let binary = "/usr/bin/gcc";
9   let output_file = "output";
10  let args = convert_args(vec![binary, "input.c", "-O2", "-o", output_file]);
11  execv(&CString::new(binary).unwrap(), &args);
12
13  // execute
14  let output_file = CString::new(output_file).unwrap();
15  execv(&CString::new(output_file).unwrap(), &[output_file]);
```

# 절대 유저를 믿지 말자

**유저는 개발자의 의도대로 서비스를 사용하지 않는다.**

만약 아래 코드를 컴파일하면 어떻게 될까?

```
// from https://stackoverflow.com/a/27755321

template <class T>
struct eat2
{
    using inner = eat2<eat2<T>>;
    static constexpr int value() {
        return inner::value();
    }
};

int main()
{
    eat2<int> e;
    cout << e.value() << endl;
    return 0;
}
```

# 절대 유저를 믿지 말자

유저는 이런 코드를 제출할 수도 있다.

```c
#include <stdlib.h>

int main() {
    system("shutdown -h now");
    return 0;
}
```

source: https://www.flickr.com/photos/66831573@N00/256076962

# Sandbox

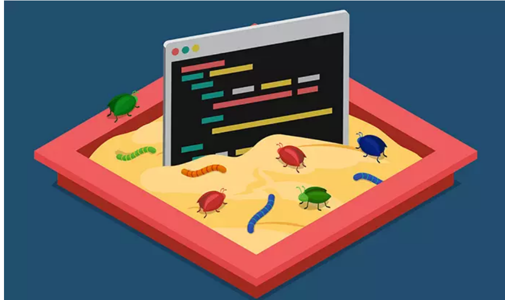
Sand + box = 모래 상자

모래가 들어있는 간이 놀이터

source: https://gty.im/1224000777

# 기술에서의 Sandbox

호스트가 안전한 테스트 환경: Sandbox 밖의 환경에 영향을 미치지 않는다.



코드를 실행해주는 많은 서비스들이 샌드박스 환경을 제공한다. 대표적으로:

- CodeSandbox
- repl.it
- tio.run
- 백준 온라인 저지
- etc...

source: https://www.braveriver.com/blog/how-sandboxing-can-help-you/

# 기술에서의 Sandbox

사용자가 악의적인 코드를 입력했을 때, 서버에 영향을 미치지 않도록 Sandbox 환경을 만들어야 한다.

가상 머신과 Docker를 고려해볼 수 있지만 이 둘은 아래와 같은 이유로 적합하지 않다.

## 가상 머신

- 보통 가상 머신은 호스트와 독립적인 머신으로 동작하므로 별도의 OS를 설치해야한다.
- 가상 머신 내부에서 프로그램 실행을 위해 가상 머신을 조작할 수 있는 별도의 API를 사용해야 한다.
- 독립적인 머신이므로 실제 OS가 동작하는 만큼의 메모리/디스크 공간이 필요하며, 스케일링이 굉장히 어렵다.

## Docker

- Docker는 호스트와 독립적인 환경을 만들 수 있지만, 컨테이너를 생성하는데 시간이 걸린다.

공통적으로, 실행되는 코드에 직접적인 제약을 걸거나, 실행 시간 제한 등을 두기 어렵다.

# seccomp

Secure Computing Mode

리눅스 커널에서 제공하는 기능. 프로세스가 사용할 수 있는 system call을 제한할 수 있다.

# libseccomp

seccomp을 쉽게 사용할 수 있도록 도와주는 라이브러리.

https://github.com/seccomp/libseccomp

# filter shutdown system call

```
1   let ctx = unsafe { seccomp_init(SCMP_ACT_ALLOW) };
2
3   let syscall_id = unsafe { seccomp_sys::seccomp_syscall_resolve_name(CString::new("shutdown").unwrap().as_ptr()) };
4
5   let rule_add_result = unsafe { seccomp_rule_add(ctx, SCMP_ACT_KILL, syscall_id, 0) };
6
7   let load_result = unsafe { seccomp_load(ctx) };
8
9   let fork_result = unsafe { nix::unistd::fork() };
10  match fork_result {
11      Ok(nix::unistd::ForkResult::Child) => {
12          // do something
13      },
14      Ok(nix::unistd::ForkResult::Parent{ child }) => {
15          // do something
16      },
17      Err(e) => {
18          panic!("Fork failed");
19      }
20  }
21
22  unsafe { seccomp_release(ctx) };
```

# filter shutdown system call

```
1    let ctx = unsafe { seccomp_init(SCMP_ACT_ALLOW) };
2
3    let syscall_id = unsafe { seccomp_sys::seccomp_syscall_resolve_name(CString::new("shutdown").unwrap().as_ptr()) };
4
5    let rule_add_result = unsafe { seccomp_rule_add(ctx, SCMP_ACT_KILL, syscall_id, 0) };
6
7    let load_result = unsafe { seccomp_load(ctx) };
8
9    let fork_result = unsafe { nix::unistd::fork() };
10   match fork_result {
11       Ok(nix::unistd::ForkResult::Child) => {
12           // do something
13       },
14       Ok(nix::unistd::ForkResult::Parent{ child }) => {
15           // do something
16       },
17       Err(e) => {
18           panic!("Fork failed");
19       }
20   }
21
22   unsafe { seccomp_release(ctx) };
```

# filter shutdown system call

```
1   let ctx = unsafe { seccomp_init(SCMP_ACT_ALLOW) };
2
3   let syscall_id = unsafe { seccomp_sys::seccomp_syscall_resolve_name(CString::new("shutdown").unwrap().as_ptr()) };
4
5   let rule_add_result = unsafe { seccomp_rule_add(ctx, SCMP_ACT_KILL, syscall_id, 0) };
6
7   let load_result = unsafe { seccomp_load(ctx) };
8
9   let fork_result = unsafe { nix::unistd::fork() };
10  match fork_result {
11      Ok(nix::unistd::ForkResult::Child) => {
12          // do something
13      },
14      Ok(nix::unistd::ForkResult::Parent{ child }) => {
15          // do something
16      },
17      Err(e) => {
18          panic!("Fork failed");
19      }
20  }
21
22  unsafe { seccomp_release(ctx) };
```

# filter shutdown system call

```
1   let ctx = unsafe { seccomp_init(SCMP_ACT_ALLOW) };
2
3   let syscall_id = unsafe { seccomp_sys::seccomp_syscall_resolve_name(CString::new("shutdown").unwrap().as_ptr()) };
4
5   let rule_add_result = unsafe { seccomp_rule_add(ctx, SCMP_ACT_KILL, syscall_id, 0) };
6
7   let load_result = unsafe { seccomp_load(ctx) };
8
9   let fork_result = unsafe { nix::unistd::fork() };
10  match fork_result {
11      Ok(nix::unistd::ForkResult::Child) => {
12          // do something
13      },
14      Ok(nix::unistd::ForkResult::Parent{ child }) => {
15          // do something
16      },
17      Err(e) => {
18          panic!("Fork failed");
19      }
20  }
21
22  unsafe { seccomp_release(ctx) };
```

# filter shutdown system call

```
1   let ctx = unsafe { seccomp_init(SCMP_ACT_ALLOW) };
2
3   let syscall_id = unsafe { seccomp_sys::seccomp_syscall_resolve_name(CString::new("shutdown").unwrap().as_ptr()) };
4
5   let rule_add_result = unsafe { seccomp_rule_add(ctx, SCMP_ACT_KILL, syscall_id, 0) };
6
7   let load_result = unsafe { seccomp_load(ctx) };
8
9   let fork_result = unsafe { nix::unistd::fork() };
10  match fork_result {
11      Ok(nix::unistd::ForkResult::Child) => {
12          // do something
13      },
14      Ok(nix::unistd::ForkResult::Parent{ child }) => {
15          // do something
16      },
17      Err(e) => {
18          panic!("Fork failed");
19      }
20  }
21
22  unsafe { seccomp_release(ctx) };
```

# filter shutdown system call

```
1   let ctx = unsafe { seccomp_init(SCMP_ACT_ALLOW) };
2
3   let syscall_id = unsafe { seccomp_sys::seccomp_syscall_resolve_name(CString::new("shutdown").unwrap().as_ptr()) };
4
5   let rule_add_result = unsafe { seccomp_rule_add(ctx, SCMP_ACT_KILL, syscall_id, 0) };
6
7   let load_result = unsafe { seccomp_load(ctx) };
8
9   let fork_result = unsafe { nix::unistd::fork() };
10  match fork_result {
11      Ok(nix::unistd::ForkResult::Child) => {
12          // do something
13      },
14      Ok(nix::unistd::ForkResult::Parent{ child }) => {
15          // do something
16      },
17      Err(e) => {
18          panic!("Fork failed");
19      }
20  }
21
22  unsafe { seccomp_release(ctx) };
```

# filter shutdown system call

```
1   let ctx = unsafe { seccomp_init(SCMP_ACT_ALLOW) };
2
3   let syscall_id = unsafe { seccomp_sys::seccomp_syscall_resolve_name(CString::new("shutdown").unwrap().as_ptr()) };
4
5   let rule_add_result = unsafe { seccomp_rule_add(ctx, SCMP_ACT_KILL, syscall_id, 0) };
6
7   let load_result = unsafe { seccomp_load(ctx) };
8
9   let fork_result = unsafe { nix::unistd::fork() };
10  match fork_result {
11      Ok(nix::unistd::ForkResult::Child) => {
12          // do something
13      },
14      Ok(nix::unistd::ForkResult::Parent{ child }) => {
15          // do something
16      },
17      Err(e) => {
18          panic!("Fork failed");
19      }
20  }
21
22  unsafe { seccomp_release(ctx) };
```

# filter shutdown system call

```
1   let ctx = unsafe { seccomp_init(SCMP_ACT_ALLOW) };
2
3   let syscall_id = unsafe { seccomp_sys::seccomp_syscall_resolve_name(CString::new("shutdown").unwrap().as_ptr()) };
4
5   let rule_add_result = unsafe { seccomp_rule_add(ctx, SCMP_ACT_KILL, syscall_id, 0) };
6
7   let load_result = unsafe { seccomp_load(ctx) };
8
9   let fork_result = unsafe { nix::unistd::fork() };
10  match fork_result {
11      Ok(nix::unistd::ForkResult::Child) => {
12          // do something
13      },
14      Ok(nix::unistd::ForkResult::Parent{ child }) => {
15          // do something
16      },
17      Err(e) => {
18          panic!("Fork failed");
19      }
20  }
21
22  unsafe { seccomp_release(ctx) };
```

# system calls

system call 목록은 아래의 man 명령어나 페이지에서 확인할 수 있다

- `man syscalls`
- https://www.man7.org/linux/man-pages/man2/syscalls.2.html
- https://linux.die.net/man/

# 온라인 저지가 동작하는 간략한 흐름

- 유저가 문제를 해결하는 프로그램의 코드를 서버에 제출한다.
- 서버는 유저가 제출한 코드를 실행/컴파일한다.
- 코드를 실행/컴파일하며, 시간/메모리 제한을 넘어서지 않는지 확인한다.
- 코드가 정답인지 확인한다.
- 결과를 유저에게 알려준다.

# setrlimit

프로세스가 사용할 수 있는 자원의 양을 제한할 수 있다.

- `RLIMIT_AS` address space
- `RLIMIT_CPU` cpu time
- ...

```
libc::setrlimit(
    [resource id],
    &libc::rlimit {
        rlim_cur: *value,
        rlim_max: *value,
    },
)
```

# setrlimit

```
libc::setrlimit(
  libc::RLIMIT_AS,
  &libc::rlimit {
      rlim_cur: 268435456, // 256MiB
      rlim_max: 268435456,
  },
)

libc::setrlimit(
    libc::RLIMIT_CPU,
    &libc::rlimit {
        rlim_cur: 1, // 1초
        rlim_max: 1,
    },
)
```

# wrap up

```
match fork_result {
    Ok(nix::unistd::ForkResult::Child) => {
      libc::setrlimit(
        libc::RLIMIT_AS,
        &libc::rlimit {
            rlim_cur: 268435456, // 256MiB
            rlim_max: 268435456,
        },
      )

      libc::setrlimit(
          libc::RLIMIT_CPU,
          &libc::rlimit {
              rlim_cur: 1, // 1초
              rlim_max: 1,
          },
      )
      execv(&path, &args);
    },
    Ok(nix::unistd::ForkResult::Parent{ child }) => {
        // ...
    },
    // ...
}
```

# 사용한 메모리, 실행 시간 측정

waitpid와 getrusage를 이용해 사용한 메모리를 측정할 수 있다.

```
1   loop {
2       let wait_result = unsafe { waitpid(child_pid, &mut status, WNOHANG) };
3       let rusage = unsafe {
4           match getrusage(RUSAGE_CHILDREN, usage.as_mut_ptr()) == 0 {
5               true => usage.assume_init(),
6               false => {
7                   panic!("getrusage failed");
8               }
9           }
10      };
11
12      // cpu user time
13      rusage.ru_utime.tv_sec as u64
14      rusage.ru_utime.tv_usec as u32
15
16      // cpu system time
17      rusage.ru_stime.tv_sec as u64
18      rusage.ru_stime.tv_usec as u32
19
20      // memory
21      rusage.ru_maxrss as u64
22
```

# 사용한 메모리, 실행 시간 측정

`waitpid`와 `getrusage`를 이용해 사용한 메모리를 측정할 수 있다.

```
loop {
    let wait_result = unsafe { waitpid(child_pid, &mut status, WNOHANG) };
    let rusage = unsafe {
        match getrusage(RUSAGE_CHILDREN, usage.as_mut_ptr()) == 0 {
            true => usage.assume_init(),
            false => {
                panic!("getrusage failed");
            }
        }
    };

    // cpu user time
    rusage.ru_utime.tv_sec as u64
    rusage.ru_utime.tv_usec as u32

    // cpu system time
    rusage.ru_stime.tv_sec as u64
    rusage.ru_stime.tv_usec as u32

    // memory
    rusage.ru_maxrss as u64
```

# 사용한 메모리, 실행 시간 측정

waitpid와 getrusage를 이용해 사용한 메모리를 측정할 수 있다.

```
loop {
    let wait_result = unsafe { waitpid(child_pid, &mut status, WNOHANG) };
    let rusage = unsafe {
        match getrusage(RUSAGE_CHILDREN, usage.as_mut_ptr()) == 0 {
            true => usage.assume_init(),
            false => {
                panic!("getrusage failed");
            }
        }
    };

    // cpu user time
    rusage.ru_utime.tv_sec as u64
    rusage.ru_utime.tv_usec as u32

    // cpu system time
    rusage.ru_stime.tv_sec as u64
    rusage.ru_stime.tv_usec as u32

    // memory
    rusage.ru_maxrss as u64
```

# 오늘 한 것

- `seccomp`로 syscall filtering
- `setrlimit`로 메모리, 시간 제한
- `getrusage`로 메모리, 시간 측정

# 더 해야할 것

- 다양한 언어 지원하기
- input/output redirection
- 채점 결과 전파
- 언어별 시간/메모리/syscall 제한
    - Python은 기본적으로 `getdents64`를 호출하므로 seccomp filter에서 제외해야 한다.

# 더 해야할 것

- 다양한 언어 지원하기
- input/output redirection
- 채점 결과 전파
- 언어별 시간/메모리/syscall 제한
  - Python은 기본적으로 `getdents64`를 호출하므로 seccomp filter에서 제외해야 한다.

**https://github.com/blurfx/sandbox**

**eof**