Operating Systems Project 1 Wiki

2018008904 이성진

Contents

- What is xv6?
- User mode & codes
- Codes for the kernel
- Other files
- getppid() implementation
- Testing with user program

1. What is xv6?

- xv6는 MIT에서 Unix의 여섯 번째 버전을 바탕으로 개발된 교육용 운영체제입니다.
- 기반이 된 Unix V6와는 다르게 x86 머신에서 동작할 수 있고, 멀티코어를 지원합니다.
- xv6는 간단한 운영체제이지만 프로세스 관리, 메모리/파일 시스템 제어 및 멀티태스킹 등 현대적인 OS가 가지고 있는 기능을 지니고 있습니다.
- xv6를 구성하고 있는 파일들은 공통 참조용 헤더 파일들과 유저 모드에서 실행될 수 있는 코드, 그리고 커널을 구성하기 위한 코드 등으로 구성되어 있습니다.

2. User mode & codes

- 기본적으로 xv6를 qemu 위에서 부팅하게 되면 쉘 프로그램이 실행되어 입력을 기다리게 됩니다.
- 다른 운영체제와 마찬가지로, 쉘 프로그램은 사용자가 명령을 입력할 때까지 대기하며 명령어가 입력되면 컴파일된 유저 프로그램들 중에 입력된 프로그램이 있는 경우 실행합니다.

```
Text string: user.h

  File         Line
0 cat.c         3 #include "user.h"
1 echo.c        3 #include "user.h"
2 forktest.c    6 #include "user.h"
3 grep.c        5 #include "user.h"
4 init.c        5 #include "user.h"
5 kill.c        3 #include "user.h"
6 ln.c          3 #include "user.h"
7 ls.c          3 #include "user.h"
8 mkdir.c       3 #include "user.h"
9 my_userapp.c  3 #include "user.h"
a printf.c      3 #include "user.h"
b rm.c          3 #include "user.h"
c sh.c          4 #include "user.h"
d stressfs.c   12 #include "user.h"
e ulib.c        4 #include "user.h"
f umalloc.c     3 #include "user.h"
g usertests.c   4 #include "user.h"
h wc.c          3 #include "user.h"
i zombie.c      6 #include "user.h"
```

[user.h 파일이 참조된 코드 목록을 cscope를 활용하여 나열한 모습]

- 다른 Unix 계열 운영체제에서 기본적으로 지원하는 **cat, echo, grep, kill, ln, ls, mkdir, rm, sh** 등의 프로그램이 구현되어 있습니다. 각각의 프로그램은 [프로그램 이름].c 파일에 정의되어 있으며, main 함수 내부에서 사용이 종료되는 부분에 **exit** 함수를 이용해 프로세스를 종료하게 되어 있습니다.
- 기본적인 프로그램 말고도 운영 체제의 프로세스나 메모리, 파일 시스템 관리 기능을 테스트할 수 있는 **forktest, zombie, stressfs** 그리고 **usertests** 등의 프로그램이 구현되어 있습니다.
- 첫 실습 시간에 구현했던 system call인 **myfunction**을 호출하는 유저 프로그램인 **my_userapp.c** 파일 역시 컴파일되어 실행 가능합니다. 이번 프로젝트에선 뒤에 구현할 **getppid**를 호출해서 테스트하기 위한 프로그램으로 **project01.c**를 새로 작성했습니다.
- 이런 유저 애플리케이션을 개발할 때 사용하는 **printf, malloc** 그리고 **free**와 같은 라이브러리 함수를 포함하는 **ulib.c, printf.c, umalloc.c** 등의 파일도 있습니다.
- 유저 모드에서 사용되는 애플리케이션과 라이브러리의 변수 및 함수들은 모두 **user.h** 파일에 선언되어 있습니다.

3. Codes for the kernel

- User mode에서 실행되는 코드를 제외한 나머지 거의 모든 부분은 커널, 즉 OS 그 자체를 구성하는 코드가 대부분입니다.
- 처음 qemu가 실행된 뒤 부팅을 하는 과정에 사용되는 코드는 **mkfs.c, bootmain.c, bootasm.S, main.c**입니다.
- filesystem을 지원하기 위해서 **fs.c, fs.h, file.c, file.h, log.c** 코드가 준비되어 있습니다.
- **mmu.h, vm.c, kalloc.c** 는 메모리 관리를 위한 코드입니다.
- **traps.h, trap.c, trapasm.S, ioapic.c, lapic.c, picirq.c**는 인터럽트 관련 코드입니다.
- system call을 관리하기 위한 **syscall.h, syscall.c** 그리고 프로세스 관련 syscall들이 구현된 **sysproc.c**와 파일 관련 **sysfile.c**가 있습니다.
- **buf.h, bio.c, console.c, ide.c, kbd.h, kbd.c, memide.c, uart.c** 등의 파일은 I/O 시스템을 구현합니다.
- **defs.h와 param.h**는 커널 모드에서 호출할 함수나 하드코딩된 단위 숫자들이 선언되어 있는 헤더 파일입니다.
- 운영체제의 프로세스 관리를 총괄하는 코드인 **proc.h, proc.c**와 실행 파일을 메모리로 올려주는 **exec.c**가 프로세스 관련 기능을 구현합니다.
- **mp.c, mp.h** 는 멀티코어 프로그래밍 관련 코드입니다.
- **proc.c**에 OS가 프로세스들을 컨트롤하기 위한 **ptable** 구조체가 정의되어 있는데, 이때 syncronization을 위한 lock이 구현되어 있는 **spinlock.h, spinlock.c, sleeplock.h, sleeplock.c**가 준비되어 있습니다.

4. Other files

- 커널/유저 모드 둘 다 사용 가능한 파일들로 **types.h, fcntl.h, stat.h** 등의 파일이 있는데, **uint, uchar**과 같은 타입 선언이나 하드코딩된 숫자 값 등의 정보가 저장되어 있는 참조용 파일들입니다.

5. getppid() implementation

- **getppid()** system call을 구현하기 위해서 비슷한 기능을 하는 **getpid**의 정의를 확인해 보았더니 현재 프로세스를 나타내는 구조체 **proc**가 존재하는 것을 확인했습니다.

```
int
sys_getpid(void)
{  return myproc()->pid;
}
```

[part of **sysproc.c**]

```
struct proc {
  uint sz;                     // Size of process memory (bytes)
  pde_t* pgdir;                // Page table
  char *kstack;                // Bottom of kernel stack for this process
  enum procstate state;        // Process state
  int pid;                     // Process ID
  struct proc *parent;         // Parent process
  struct trapframe *tf;        // Trap frame for current syscall
  struct context *context;     // swtch() here to run process
  void *chan;                  // If non-zero, sleeping on chan
  int killed;                  // If non-zero, have been killed
  struct file *ofile[NOFILE];  // Open files
  struct inode *cwd;           // Current directory
  char name[16];               // Process name (debugging)
};
```

[part of **proc.h**]

- proc.h에 있는 프로세스 구조체 정의와 sysproc.c에 정의된 sys_getpid 함수를 참조하여 현재 프로세스의 부모 프로세스에 접근한 뒤 해당 구조체의 pid 값을 반환하도록 하는 getppid() 함수를 sysproc.c에 새롭게 정의했습니다.

```
int
sys_getppid(void)
{  return myproc()->parent->pid;
}
```

[part of **sysproc.c**]

- 새로 추가된 sys_getppid가 정상적으로 작동하게 하도록 syscall.h와 syscall.c의 system call 목록에 sys_getppid를 추가하였습니다.

```
Text string: sys_getppid

  File      Line
0 syscall.c 107 extern int sys_getppid(void);
1 syscall.c 132 [SYS_getppid] sys_getppid,
2 syscall.h  24 #define SYS_getppid 23
3 sysproc.c  94 sys_getppid(void)
```

[sys_getppid를 다른 system call처럼 실행되도록 추가한 모습]

- 이후 유저 프로그램에서 해당 system call을 호출할 수 있도록, usys.S에 매크로를 등록하고 user.h에 getppid의 선언을 입력해서 user.h를 참조하면 getppid system call을 활용할 수 있도록 했습니다.

```
C symbol: getppid

  File     Function Line
0 user.h   exit      27 int getppid(void );
1 usys.S   SYSCALL   33 SYSCALL(getppid)
```

[sys_getppid를 다른 system call처럼 실행되도록 추가한 모습]

6. Testing with user program

- 새로운 유저 앱 project01을 만들고 getpid()와 getppid()를 호출하여 현재 프로세스의 pid와 ppid를 출력하도록 합니다.
- Makefile의 유저 애플리케이션 목록에 project01을 추가하여 컴파일한 후 쉘에서 실행할 수 있도록 합니다.

```
#include "types.h"
#include "stat.h"
#include "user.h"

// A user program which print current
// process' pid & parent process' pid.

int main(){

        printf(1, "My pid is %d\n", getpid());
        printf(1, "My ppid is %d\n", getppid());
    exit();

    return 0; // This part will not be reached.
}
```

[**project01.c**]

```
UPROGS=\
        _cat\
        _echo\
        _forktest\
        ...
        _zombie\
        _my_userapp\
        _project01\
```

```
EXTRA=\
        mkfs.c ulib.c user.h cat.c echo.c forktest.c
grep.c kill.c\
        ln.c ls.c mkdir.c rm.c stressfs.c usertests.c
wc.c zombie.c\
        printf.c umalloc.c my_userapp.c project01.c\
        README dot-bochsrc *.pl toc.* runoff runoff1
runoff.list\
        .gdbinit.tmpl gdbutil\
```

[part of **Makefile**]

- make 명령어를 실행하여 xv6를 컴파일한 뒤 부팅한 다음 쉘 프로그램에 project01을 입력하면 project01 프로그램이 실행되고 현재 프로세스의 pid와 ppid를 출력하게 됩니다.

```
Booting from Hard Disk..xv6...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ project01
My pid is 3
My ppid is 2
$
```

- 현재 쉘에서 project01을 반복적으로 실행하면 새로운 프로세스가 생성되기 때문에 pid가 계속 증가하지만, 부모 프로세스인 쉘은 그대로 실행 중이기 때문에 ppid는 변하지 않는 것을 확인할 수 있습니다.

```
$ project01
My pid is 4
My ppid is 2
$ project01
My pid is 5
My ppid is 2
$
```

- 만약 새로운 쉘을 다시 시작하도록 하는 init이나 sh 프로그램을 실행시킨 후 project01을 실행하면 project01을 fork한 부모 프로세스가 달라졌기 때문에 ppid 역시 달라진 것을 알 수 있습니다.
- 새로 생성된 쉘 프로세스들을 kill 한 뒤 다시 project01을 실행시키면 가장 처음 실행 중이던 쉘의 pid가 ppid로 다시 나타나는 것을 알 수 있습니다.

```
init: starting sh
$ project01
My pid is 3
My ppid is 2
$ init
init: starting sh
$ project01
My pid is 6
My ppid is 5
$ sh
$ project01
My pid is 8
My ppid is 7
$ kill 4
$ $ project01
My pid is 10
My ppid is 2
$
```