Assignment1 wiki

2020064257 이준현

- Design

Kernal

system call에 function을 등록하여 kernel이 처리할 수 있게 함

1. system call과 관련된 process 관련 기능을 구현하는 kernel/syspro.c에 부모 프로세스의 pid를 반환하는 getppid function 정의
2. syscall.h에 SYS_getppid의 system call number를 등록하고 syscall.c의 system call table에 SYS_getppid를 등록

User

system call을 등록하여 User가 kernel에서 제공하는 system call을 사용할 수 있게 함

1. user/user.h에 getppid function 선언 추가
2. 시스템 호추 관련 코드(usys.S)를 자동으로 생성해주는 Perl script user/usys.pl에 getppid entry 추가

user program에서 wrapper function으로 system call을 호출할 수 있게 함

ppid.c

user directory에 ppid.c를 만들어, printf함수를 이용해 원하는 동작(student ID, process ID, parent process ID 출력)을 하는 user program 작성, 이때 이미 있는 getpid function과 새로 등록한 getppid function을 이용

- Implementation

Kernel

1. kernel/proc.c에 새로운 함수 sys_getppid(void) 정의

```
uint64
sys_getppid(void)
{
  return myproc()->parent->pid;
}
```

cpu가 다루고 있는 process 구조체를 반환하는, kernel/proc.c에 있는 myproc()

```
// Return the current struct proc *, or zero if none.
struct proc*
myproc(void)
{
  push_off();
  struct cpu *c = mycpu();
  struct proc *p = c->proc;
  pop_off();
  return p;
}
```

을 활용하여 그 structure에서 그 member인 parent process structure에 접근한 뒤, 또 그 member인 pid를 반환하는 함수, 실행시키면 ppid를 반환받게 된다.

2. kernel/syscall.h에 “#define SYS_getppid 22”
줄을 추가하여 SYS_getppid를 system call number를 22로 등록
3. kernel/syscall.c에 “extern uint64 sys_getppid(void);”
줄을 추가하여 다른 파일에 sys_getppid함수가 정의돼 있음을 명시한 뒤,
static uint64 (*syscalls[](void) 에 “[SYS_getppid] sys_getppid,”
줄을 추가하여 새로 등록한 system call number와 function을 mapping

User

1. user/user.h에 “int getppid(void);” 줄을 추가하여 getppid wrapper function의 선언 추가
2. user/usys.pl에 entry(“getppid”); 줄을 추가

usys.pl은 syscall.h파일을 읽어 system call number와 이름을 파싱
파싱된 system call 정보를 기반으로 usys.S파일을 생성, sys_getppid에 대한 wrapper function을 assembly로 생성

```
.global getppid
getppid:
 li a7, SYS_getppid
 ecall
 ret
```

가 usys.S에 생성, wrapper function는 system call number를 a7 register에 로드하고 ecall 명령어를 호출하여 system call을 실행

ppid.c

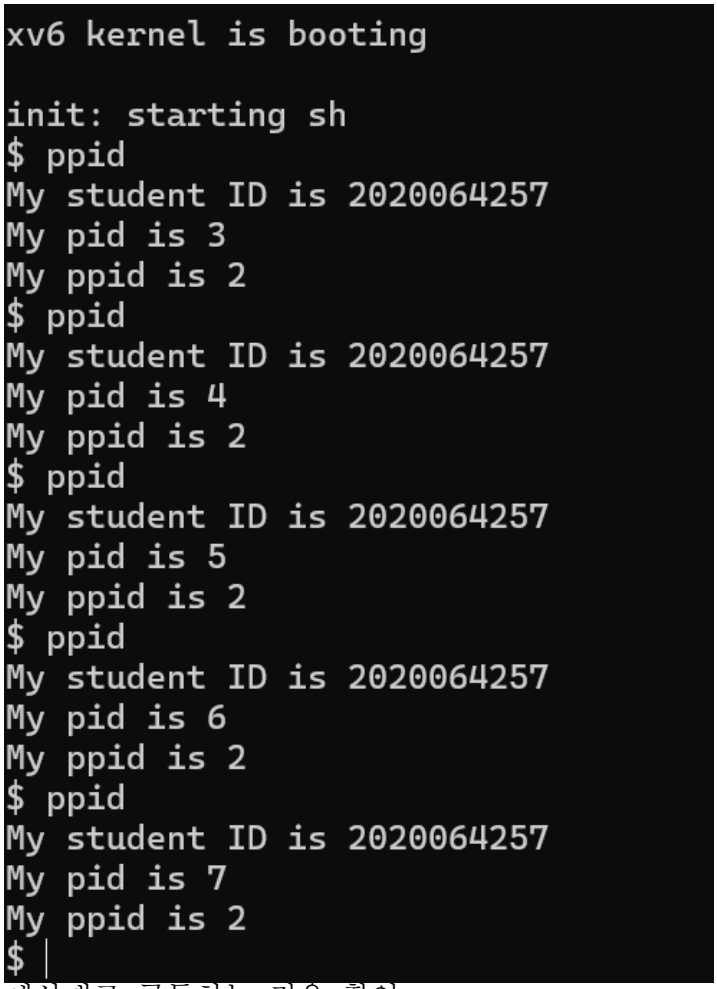
```c
#include "kernel/types.h"
#include "kernel/stat.h"
#include "user/user.h"

int
main(int argc, char *argv[])
{
        printf("My student ID is 2020064257\nMy pid is %d\nMy ppid is %d\n", getpid(), getppid());
        exit(0);
}
```

실행하면 printf()를 이용하여 단순히, student ID, pid, ppid를 출력, pid는 getpid()에서 반환, ppid는 새로 구현한 getppid()에서 반환

- Results

make qemu를 이용하여 xv6을 booting한 뒤, shell에 ppid를 입력하면,
ppid.c가 실행되는데, getpid()와 getppid()를 통해 system call을 호출하고 kernal이 해당 system call을 실행하여 해당 process의 pid와 parent pid를 반환받아 printf를 통해 출력
ppid는 해당 process의 parent인 shell의 ID가 출력, init process가 PID를 가지고 shell process는 PID 2를 가질 것이므로 ppid는 2가 출력되고, pid는 해당 프로세스를 실행할 때마다 3에서 하나씩 증가할 것으로 예상



예상대로 구동하는 것을 확인

- Troubleshooting

parent process를 다루는 일에서 lock을 설정한 뒤에 실행해야 하지 않나 하는 의문점이 생겼는데, 찾아본 결과 myproc()함수는 현재 CPU에서 실행 중인 process proc structure를 반환하며, getppid가 실행되는 동안 다른 process가 현재 process의 proc structure을 수정할 가능성이 낮고, int또는 uint64와 같은 기본 자료형인 ppid를 읽어 오기만 하기 때문에 원자적으로 수행되어 lock을 굳이 설정할 필요가 없다는 답을 얻음.