Assignment wiki

> 2021011158 김선희

📌 요구사항 분석

- getppid() system call 구현
- getppid() system call을 호출하는 user program 이름은 ppid여야 함

Design

1. getpid() system call 분석

```
getpid:
 li a7, SYS_getpid
 ecall
 ret
.global sbrk
```

```
[SYS_getpid]  sys_getpid,
```

- getpid는 sys_getpid system call로 매칭되므로, 해당 system call의 정의를 들여다보고자 한다.

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

```
struct proc*    myproc();
```

```
// Per-process state
struct proc {
  struct spinlock lock;

  // p->lock must be held when using these:
  enum procstate state;        // Process state
  void *chan;                  // If non-zero, sleeping on chan
  int killed;                  // If non-zero, have been killed
  int xstate;                  // Exit status to be returned to parent's wait
  int pid;                     // Process ID

  // wait_lock must be held when using this:
  struct proc *parent;         // Parent process

  // these are private to the process, so p->lock need not be held.
  uint64 kstack;               // Virtual address of kernel stack
  uint64 sz;                   // Size of process memory (bytes)
  pagetable_t pagetable;       // User page table
```

```
  struct trapframe *trapframe; // data page for trampoline.S
  struct context context;      // swtch() here to run process
  struct file *ofile[NOFILE];  // Open files
  struct inode *cwd;           // Current directory
  char name[16];               // Process name (debugging)
};
```

- sys_getpid는 type이 proc 구조체인 myproc() 객체의 멤버 변수 pid에 접근하여 process id를 가져온다.

2. 구현 아이디어

- proc 구조체를 보면, pid 뿐만 아니라 parent를 멤버 변수로 가진다.

⇒ ppid는 parent의 pid이므로, `myproc()→parent→pid` 를 통해 parent process id에 접근할 수 있을 것이다.

3. xv6의 system call 처리 방식 분석

user space

1. user space에 정의된 c 파일에서 system call을 호출한다.

   ex) cat.c

   ```
   while((n = read(fd, buf, sizeof(buf))) > 0) {
   ```

2. user.h에 해당 system call이 선언되어 있어야 하고, 이것에 대한 정의는 usys.S에 있다. 여기서 usys.S에 정의된 대로 a7 레지스터에 저장된 system call 번호를 가지고 ecall을 통해 kernel space로 들어간다. usys.S는 사용자가 직접 만드는 것이 아니고, usys.pl에 system call 이름을 넣으면, 그에 대한 usys.S를 만들어준다.

```
// system calls
int read(int, void*, int);
```

```
.global read
read:
 li a7, SYS_read
 ecall
 ret
```

kernel space

3. syscall.c에서 해당 system call과 일치하는 system call을 찾아 그 코드를 실행한다.
4. 해당 system call의 선언은 defs.h에, 정의는 sysfile.c, prac_syscall.c 등 system call 함수를 정의해 둔 c 파일에 있다. 그리고 만든 c 파일에 대한 o 파일을 만들기 위해 makefile에 (파일이름).o를 추가해야 한다.

4. 구현 방식 구상

kernel space

1. parent process id값을 리턴하는 `sys_getppid()` 를 구현한다.
2. 새로 정의한 system call을 `syscall.h` 와 `syscall.c` 에 추가한다.

user space

3. 사용할 프로그램인 `ppid.c` 를 작성한다.
4. `user.h` 에 `getppid()` 를 추가하고, 이에 해당하는 매크로를 생성하기 위해 `usys.pl` 에 함수명을 추가한다.

Implementation

kernel space

1. getpid()를 선언한다.
   - defs.h에 interface 역할을 할 getpid()를 선언한다.

```
int             getid(void);
```

2. sys_getpid()를 구현한다.
   - 나는 실습 시간에 만들어 둔 prac_syscall.c에 다음과 같이 정의하였다.
   - 먼저 현재 프로세스 객체인 myproc()의 parent 멤버 변수에 접근하여, 그의 pid를 가져오는 식으로 구현했다.
   - 관련 header도 함께 추가했다.

```
#include "param.h"
#include "memlayout.h"
#include "spinlock.h"
#include "proc.h"
...
// assignment1
// system call: get parent process id
int sys_getppid(void){
    return myproc()->parent->pid;
}
```

3. 정의한 system call을 사용하기 위한 정의를 추가한다.
   - syscall.h에 system call에 대한 index 선언을 했다.
   - syscall.c에 syscall 내에서 a7 레지스터에 넘겨주기 위한 배열 (wrapper function)에 getpid system call을 추가했다.

```
#define SYS_getppid    23
```

```
extern uint64 sys_getppid(void);
...
[SYS_getppid]   sys_getppid,
```

user space

1. 주어진 요구사항을 출력할 user program인 ppid.c를 구현한다.
   - 이미 구현되어 있는 getpid()와 kernel space에 직접 구현한 getppid()를 호출하여 process id와 parent process id 값을 각각 불러온다.

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

int main(int argc, char *argv[]) {
    fprintf(2, "My student ID is 2021011158\n");
    int pid_val = getpid();
    fprintf(1, "My pid is %x\n", pid_val);
    int ppid_val = getppid();
    fprintf(1, "My ppid is %x\n", ppid_val);
    exit(0);
};
```

2. getppid()를 user program에서 사용할 수 있도록 헤더파일과 매크로에 추가한다.

```
int getppid(void);
```

```
entry("getppid");
```

Makefile

실행파일 생성을 위해 makefile에 새로 만든 user program과 kernel program을 추가한다.

```
$K/prac_syscall.o  \
...
$U/_ppid\
```

Result