Project 1

Report

2017314331

김도엽

Report 1

0. Project1 PDF에 주어진 조건에 부합하는 getnice, setnice, ps system call을 xv6에 추가하는 작업을 거쳤습니다. 이하 내용은 추가된 Code, Code에 대한 설명, 그리고 Result 순으로 작성될 예정입니다.

1 . 추가된 Code

1-1. getnice

1) syscall.h와 syscall.c에 추가된 Line.

```
syscall.h

#define SYS_getnice 23

syscall.c

extern int sys_getnice(void);
[SYS_getnice]   sys_getnice,
```

syscall.h에서 getnice에 대한 System call 번호 추가

syscall.c에 getnice System call 추가

2) defs.h 및 user.h에서 getnice 함수 선언

```
int             getnice(int);
```

이 line은 프로세스 ID(PID)를 나타내는 정수 인수를 사용하고 해당 프로세스의 nice 값을 반환하는 getnice 함수를 선언합니다.

3) sysproc.c에서 sys_getnice 함수 구현:

```
int
sys_getnice(void)
{
  int pid;

  if(argint(0, &pid) < 0)
    return -1;

  return getnice(pid);
}
```

sys_getnice( ) 함수는 user program에서 Process ID를 검색하고 PID로 실제 getnice 함수를 호출하는 역할을 합니다. 대상 프로세스의 nice value를 return하거나 PID 검색에 실패하면 -1을 return 합니다.

4) proc.c에 getnice 함수 추가:

```
int
getnice(int pid)
{
  struct proc *p;
  int nice = -1;

  acquire(&ptable.lock);
  for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
    if(p->pid == pid){
      nice = p->nice;
      break;
    }
  }
  release(&ptable.lock);

  return nice;
}
```

getnice 함수는 지정된 PID가 있는 proces를 찾기 위해 프로세스 테이블을 Searching합니다. 주어진 PID를 가진 process가 발견되면 함수는 해당 nice 값을 반환합니다. 프로세스를 찾을 수 없으면 함수는 -1을 반환합니다.

5) User Program 추가를 위한 getnice.c 추가 :

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

int
main(int argc, char *argv[])
{
    if(argc != 2){
        printf(2, "Usage: getnice [pid]\n");
        exit();
    }

    int pid = atoi(argv[1]);
    int nice = getnice(pid);

    if (nice >= 0) {
```

```
        printf(1, "Nice value of process %d: %d\n", pid, nice);
    } else {
        printf(2, "Error : No process corresponding to the pid or the
nice value is invalid.\n");
    }

    exit();
}
```

올바른 개수의 argv가 제공되었는지 확인합니다. getnice 시스템 호출은 pid를 인수로 사용하여 호출되며 반환 값(nice 값)은 nice 변수에 저장됩니다. 반환된 nice 값이 0보다 크거나 같으면 프로그램은 주어진 PID를 사용하여 프로세스의 nice 값을 print합니다. 반환된 nice 값이 0보다 작으면 오류 메시지 `"Error : No process corresponding to the pid or the nice value is invalid.\n"`를 print 합니다.

6) getnice를 실행한 결과 :

```
$ ps
name     pid     state           priority
init     1       SLEEPING                20
sh       2       SLEEPING                20
ps       3       RUNNING                 20
$
$ getnice 1
Nice value of process 1: 20
$ getnice 2
Nice value of process 2: 20
```

1-2. setnice

1) syscall.h와 syscall.c에 추가된 Line.

```
syscall.h

#define SYS_setnice 24


syscall.c

extern int sys_setnice(void);

[SYS_setnice]   sys_setnice,
```

setnice 시스템 호출에 고유한 번호(24)를 할당합니다. setnice 시스템 호출 번호(SYS_setnice)를 커널의 구현 함수(sys_setnice)와 연결합니다.

2) defs.h 및 user.h에서 setnice 함수 선언

```
int             setnice(int, int);
```

이 Line 은 각각 프로세스 ID(PID)와 새 nice 값을 나타내는 두 개의 정수 인수를 사용하는 setnice 함수를 선언합니다. 성공하면 0을, 실패하면 0이 아닌 값을 반환합니다.

3) sysproc.c에서 sys_ps 함수 구현:

```
int
sys_setnice(void)
{
  int pid, value;

  if(argint(0, &pid) < 0 || argint(1, &value) < 0)
    return -1;

  if (value < 0 || value > 39) {
    return -1; // Invalid nice value
  }

  return setnice(pid, value);
}
```

sys_setnice function은 인수를 사용자 프로그램에서 PID 및 새 nice 값으로 실제 setnice 함수를 호출합니다. 성공 시 0을 반환하고, 인수 검색이 실패하거나 nice 값이 유효하지 않은 경우 0이 아닌 값을 반환합니다.

4) proc.c에 setnice 함수 추가 :

```
int
setnice(int pid, int value)
{
  struct proc *p;
  int success = -1;

  acquire(&ptable.lock);
  for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
    if(p->pid == pid){
      p->nice = value;
      success = 0;
      break;
    }
  }
  release(&ptable.lock);

  return success;
}
```

setnice 함수는 입력된 PID를 가진 Process를 찾기 위해 Process Table을 반복 Searching 합니다. 주어진 PID를 가진 Process가 발견되고 새로운 nice 값이 유효한 범위(0 ~ 39) 내에 있으면 함수는 Process의 사용자가 shell에서 입력한 nice 값을 설정하고 0을 return합니다. Process를 찾을 수 없거나 새 nice 값이 유효하지 않으면 함수는 -1을 return합니다.

5) User Program 추가를 위한 setnice.c 추가 :

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

int
main(int argc, char *argv[])
{
    if(argc != 3){
        printf(2, "Usage: setnice [pid] [value]\n");
        exit();
    }

    int pid = atoi(argv[1]);
    int value = atoi(argv[2]);

    if (setnice(pid, value) == 0) {
        printf(1, "Set nice value : process %d to %d\n", pid, value);
    } else {
        printf(2, "Error: Unable to set nice value, check PID and nice
value range (0-39)\n");
    }

    exit();
}
```

올바른 개수의 argv가 제공되었는지 확인합니다. PID 및 value를 인수로 사용하여 setnice System call이 호출됩니다. System call은 주어진 PID를 가진 Process의 nice 값을 지정된 값으로 설정합니다. 그런 다음 nice 값이 변경되었음을 나타내는 메시지를 print합니다. 지정된 PID를 가진 Process가 존재하지 않거나 nice 값이 유효하지 않는다면, 0이 아닌 값을 반환하고 오류 메시지를 print 합니다.

6) setnice를 실행한 결과 :

```
$ setnice 1 30
Set nice value : process 1 to 30
$ getnice 1
Nice value of process 1: 30
$ setnice 2 10
Set nice value : process 2 to 10
$ getnice 2
Nice value of process 2: 10
```

1-3. Ps

1) syscall.h와 syscall.c에 추가된 Line.

```
syscall.h

#define SYS_ps 25

syscall.c

extern int sys_ps(void);
[SYS_ps]        sys_ps,
```

setnice 시스템 호출에 고유한 번호(25)를 할당합니다. ps 시스템 호출 번호(SYS_ps)를 커널의 구현 함수(sys_ps)와 연결합니다.

2) defs.h 및 user.h에서 ps 함수 선언

```
void            ps(int);
```

ps함수는 return value가 존재하지 않습니다. ps 함수의 주요 목적은 제공된 PID를 기반으로 Process 정보를 인쇄하는 것이며 작업 완료 후 특정 값을 반환하지 않습니다.

3) sysproc.c에서 sys_ps 함수 구현:

```
int
sys_ps(void)
{
  int pid;

  if(argint(0, &pid) < 0)
    return -1;

  ps(pid);

  return 0;
}
```

argint(0, &pid)를 호출하여 사용자 프로그램이 전달한 첫 번째(0 번째) 정수 인수(Process ID 여야 함)를 가져와 PID 변수에 저장합니다. 'argint'가 음수 값을 return 하면 오류를 나타내며 return 됩니다. 가져온 PID 값을 인수로 전달하여 ps(pid)

함수를 호출합니다. proc.c 에 구현된 ps 함수는 제공된 PID 를 기반으로 프로세스 정보를 인쇄합니다.

4) proc.c에 ps 함수 추가 :

```
void
ps(int pid)
{
  struct proc *p;

  // Print the header for the output format
  cprintf("name\t pid\t state\t\t priority\n");

  acquire(&ptable.lock);
  for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
    if(pid == 0 || p->pid == pid){
      if(p->state == UNUSED)
        continue;

      // Print process information in the desired format
      cprintf("%s\t %d\t ", p->name, p->pid);

      switch(p->state) {
        case SLEEPING:
          cprintf("SLEEPING\t\t ");
          break;
        case RUNNABLE:
          cprintf("RUNNABLE\t\t ");
          break;
        case RUNNING:
          cprintf("RUNNING\t\t ");
          break;
        case ZOMBIE:
          cprintf("ZOMBIE\t\t ");
          break;
        default:
          cprintf("UNKNOWN\t\t ");
          break;
      }

      cprintf("%d\n", p->nice);
    }
  }
  release(&ptable.lock);
}
```

5) User Program 추가를 위한 ps.c 추가 :

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

int
main(int argc, char *argv[])
{
    int pid = 0;
    if(argc == 2){
        pid = atoi(argv[1]);
    } else if (argc > 2) {
        printf(2, "Usage: ps [pid]\n");
        exit();
    }

    ps(pid);

    exit();
}
```

먼저 올바른 개수의 인수가 입력되었는지 확인합니다. 그렇지 않은 경우 사용법 메시지를 인쇄하고 종료합니다. 올바른 수의 인수가 제공되면 두 번째 인수(argv[1])를 정수(PID)로 변환하고 해당 PID로 ps System Call을 호출합니다. System Call이 실행된 후 프로그램이 종료됩니다.

6) ps를 실행한 결과 :

```
==Welcome to XV6, revised by DoYeop==
$ ps
name     pid     state           priority
init     1       SLEEPING                20
sh       2       SLEEPING                20
ps       3       RUNNING                 20
$
```

```
$ ps
name     pid     state           priority
init     1       SLEEPING                30
sh       2       SLEEPING                10
ps       11      RUNNING                 10
$ ps 1
name     pid     state           priority
init     1       SLEEPING                30
```

2. 전체 결과

```
SeaBIOS (version 1.13.0-1ubuntu1.1)


iPXE (http://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1FF8CA10+1FECCA10 CA00



Booting from Hard Disk..xv6...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
Student ID : 2017314331
Student Name : DoYeop Kim
==Welcome to XV6, revised by DoYeop==
$ ps
name     pid     state           priority
init     1       SLEEPING                20
sh       2       SLEEPING                20
ps       3       RUNNING                 20
$
$ getnice 1
Nice value of process 1: 20
$ getnice 2
Nice value of process 2: 20
$ setnice 1 30
Set nice value : process 1 to 30
$ getnice 1
Nice value of process 1: 30
$ setnice 2 10
Set nice value : process 2 to 10
$ getnice 2
Nice value of process 2: 10
$ ps
name     pid     state           priority
init     1       SLEEPING                30
sh       2       SLEEPING                10
ps       11      RUNNING                 10
$
```