컴퓨터학부 20160548 이승현 나반

1. 과제 개요

xv6를 설치, 실행하고, 여러 시스템 콜을 추가한다. 추가한 시스템 콜을 이용하여 기존의 xv6 스케줄러를 수정해 우선순위 기반의 RR 스케줄러를 구현한다.

구현 방법

- 1-(1). hello() 시스템 호출 추가.

다음과 같은 과정을 거쳐 새로운 시스템 콜을 추가하였다

- syscall.h

  #define SYS_hello 22 추가

- syscall.c

  extern int do_hello(void); 선언

  [SYS_hello] do_hello, 추가

- sysproc.c

  int do_hello(void) 구현

  cprintf()를 이용해 "helloxv6₩n" 문자열을 출력하는 함수 do_hello()를 구현

- usys.S

  SYSCALL(hello) 추가

- user.h

  int hello(void); 선언

- 1-(2). hello_name() 시스템 호출 추가.

다음과 같은 과정을 거쳐 새로운 시스템 콜을 추가하였다

- syscall.h

  #define SYS_hello_name 23 추가

- syscall.c

  extern int do_hello_name(void); 선언

  [SYS_hello_name] do_hello_name, 추가

- sysproc.c

  int do_hello_name(void) 구현

  전달받은 인자를 argstr()함수를 이용해 얻어낸 뒤, 그 문자열을 cprintf()를 이용해 출력하는 함수 do_hello_name()를 구현.

- usys.S

  SYSCALL(hello_name) 추가

- user.h

  int hello_name(const char *); 선언

- 1-(3). get_num_proc() 시스템 호출 추가.

다음과 같은 과정을 거쳐 새로운 시스템 콜을 추가하였다

- syscall.h

  #define SYS_get_num_proc 24 추가

- syscall.c

  extern int do_get_num_proc(void); 선언

  [SYS_get_num_proc] do_get_num_proc, 추가

- defs.h

  int get_num_proc(void); 선언

- proc.c

  int get_num_proc(void) 구현

  ptable을 이용해 실행중인 프로세스의 개수를 구해 반환하는 get_num_proc 함수를 구현.

- sysproc.c

  int do_get_num_proc(void) 구현

  proc.c에 정의된 get_num_proc()을 호출하고, 그 결과를 리턴하는 함수 do_get_num_proc(void)를 구현.

- usys.S

  SYSCALL(get_num_proc) 추가

- user.h

  int get_num_proc(void); 선언

- 1-(4). get_max_pid() 시스템 호출 추가.

다음과 같은 과정을 거쳐 새로운 시스템 콜을 추가하였다

- syscall.h

  #define SYS_get_max_pid 25 추가

- syscall.c

  extern int do_get_max_pid(void); 선언

  [SYS_get_max_pid] do_get_max_pid, 추가

- defs.h

  int get_max_pid(void); 선언

- proc.c

  int get_max_pid(void) 구현

  ptable에서 실행중인 모든 프로세스를 확인하여, 최대 pid 값을 찾아 리턴하는 함수 get_max_pid(void) 구현.

- sysproc.c

  int do_get_max_pid(void)

  proc.c에 정의된 get_max_pid()을 호출하고, 그 결과를 리턴하는 함수 do_get_max_pid(void)를 구현.

- usys.S

  SYSCALL(get_max_pid) 추가

- user.h

  int get_max_pid(void); 선언

- 1-(5). get_proc_info(pid, *processInfo) 시스템 호출 추가.

다음과 같은 과정을 거쳐 새로운 시스템 콜을 추가하였다

- syscall.h

  #define SYS_get_proc_info 26 추가

- syscall.c

  extern int do_get_proc_info(void); 선언

  [SYS_get_proc_info] do_get_proc_info, 추가

- defs.h

  int get_proc_info(int, struct processInfo *) 선언

- proc.c

  int get_proc_info(int pid, struct processInfo *pInfo) 구현

  인자로 받은 pid를 프로세스 id로 갖는 프로세스가 있다면 해당 프로세스의 정보를 인자로 받은 processInfo 구조체에 담고 0을 리턴하며 함수 종료. 전달받은 pid에 해당하는 프로세스를 찾지 못했다면 -1 리턴하는 함수 get_proc_info() 구현.

- sysproc.c

  int do_get_proc_info(void) 구현

  인자로 전달된 pid와 struct processInfo *를 각각 argint(), argptr()을 이용해 받은 뒤, 이 인자들을 이용하여 proc.c에 정의된 get_proc_info() 함수를 호출하고, 그 값을 그대로 리턴하는 함수 do_get_proc_info()를 구현.

- usys.S

  SYSCALL(get_proc_info) 추가

- user.h

  int get_proc_info(int, struct processInfo *); 선언

- 2. set_prio(), get_prio() 시스템 콜 추가

다음과 같은 과정을 거쳐 새로운 시스템 콜을 추가하였다

- syscall.h

  #define SYS_set_prio 27 추가

  #define SYS_get_prio 28 추가

- syscall.c

  extern int do_set_prio(void); 선언

  extern int do_get_prio(void); 선언

  [SYS_set_prio] do_set_prio, 추가

  [SYS_get_prio] do_get_prio, 추가

- defs.h

  int set_prio(int); 선언

  int get_prio(void); 선언

- proc.h

  struct proc { - 새로운 멤버 변수 추가

  unsigned int prio; // 스케줄링 우선순위

  int count; // 우선순위 기반 RR 스케줄링에 사용할 count

  };

- proc.c

  int set_prio(int n) 구현

  int get_prio() 구현

  struct proc에 새롭게 추가한 멤버변수인 prio의 값을 새롭게 지정하거나 찾아서 반환하는 함수 set_prio()와 get_prio()를 구현.

- sysproc.c

  int do_set_prio() 구현

  int do_get_prio() 구현

  proc.c에 정의된 set_prio()와 get_prio()를 호출하고 반환 받은 값을 그대로 리턴 하는 함수 do_set_prio(), do_get_prio() 구현

- usys.S

  SYSCALL(set_prio) 추가

  SYSCALL(get_prio) 추가

- user.h

  int set_prio(int); 선언

  int get_prio(void); 선언

- 3. 우선순위 기반 RR 스케줄러 구현.

우선순위 기반 RR 스케줄링에 사용하기 위해 proc.h의 struct proc에 count라는 멤버변수를 추가하였다. 이 변수는 int형 변수이고, 초기값은 2000이다. 이 count 값과 prio를 이용하여 우선순위 기반 RR 스케줄러를 proc.c의 scheduler()에 구현하였다. context switch가 일어날 때마다 실행 중이던 프로세스의 count 값을 모든 RUNNABLE 상태의 프로세스들의 prio(우선순위)값의 평균 값만큼 감소시킨다. RUNNABLE 상태였던 다른 프로세스들의 count값은 "(MAX_COUNT - p->count) / 2" 만큼 증가시켜 반감 효과가 일어나도록 했다. 이후 RUNNABLE 상태의 프로세스의 count값과 prio의 합을 계산하고, 이 값이 가장 큰 프로세스를 context switch해 실행되도록 한다. 이 과정을 반복해 스케줄링 하면 우선순위에 기반한 RR 스케줄링이 된다. 주의할 점은 count의 최댓값을 2000으로 설정했기 때문에 프로세스의 prio가 1000 이하의 값이 되도록 설정하는 것이 좋다.

2. 결과

1-(1) hello() 시스템 콜 추가

```
echo            2 4 12892
forktest        2 5 8328
grep            2 6 15760
init            2 7 13476
kill            2 8 12944
ln              2 9 12844
ls              2 10 15032
mkdir           2 11 13024
rm              2 12 13004
sh              2 13 23492
stressfs        2 14 13672
usertests       2 15 56376
wc              2 16 14420
zombie          2 17 12668
hello_test      2 18 12636
helloname_test 2 19 12824
getnumproc_tes 2 20 12736
getmaxpid_test 2 21 12788
getprocinfo_te 2 22 13188
seqinc_prio     2 23 15724
seqdec_prio     2 24 15708
console         3 25 0
$ hello_test
helloxv6
```

1-(2) hello_name() 시스템 콜 추가

```
init            2 7 13476
kill            2 8 12944
ln              2 9 12844
ls              2 10 15032
mkdir           2 11 13024
rm              2 12 13004
sh              2 13 23492
stressfs        2 14 13672
usertests       2 15 56376
wc              2 16 14420
zombie          2 17 12668
hello_test      2 18 12636
helloname_test 2 19 12824
getnumproc_tes 2 20 12736
getmaxpid_test 2 21 12788
getprocinfo_te 2 22 13188
seqinc_prio     2 23 15724
seqdec_prio     2 24 15708
console         3 25 0
$ hello_test
helloxv6
$ helloname_test shlee
hello shlee
$ getnumproc_test
```

1-(3) get_num_proc() 시스템 콜 추가

```
kill            2 8 12944
ln              2 9 12844
ls              2 10 15032
mkdir           2 11 13024
rm              2 12 13004
sh              2 13 23492
stressfs        2 14 13672
usertests       2 15 56376
wc              2 16 14420
zombie          2 17 12668
hello_test      2 18 12636
helloname_test 2 19 12824
getnumproc_tes 2 20 12736
getmaxpid_test 2 21 12788
getprocinfo_te 2 22 13188
seqinc_prio     2 23 15724
seqdec_prio     2 24 15708
console         3 25 0
$ hello_test
helloxv6
$ helloname_test shlee
hello shlee
$ getnumproc_test
Total Number of Active Processes: 3
```

1-(4) get_max_pid() 시스템 콜 추가

```
ls              2 10 15032
mkdir           2 11 13024
rm              2 12 13004
sh              2 13 23492
stressfs        2 14 13672
usertests       2 15 56376
wc              2 16 14420
zombie          2 17 12668
hello_test      2 18 12636
helloname_test 2 19 12824
getnumproc_tes 2 20 12736
getmaxpid_test 2 21 12788
getprocinfo_te 2 22 13188
seqinc_prio     2 23 15724
seqdec_prio     2 24 15708
console         3 25 0
$ hello_test
helloxv6
$ helloname_test shlee
hello shlee
$ getnumproc_test
Total Number of Active Processes: 3
$ getmaxpid_test
Maximum PID: 7
```

1-(5) get_proc_info(pid, *processInfo) 시스템 콜 추가

```
wc              2 16 14420
zombie          2 17 12668
hello_test      2 18 12636
helloname_test 2 19 12824
getnumproc_tes 2 20 12736
getmaxpid_test 2 21 12788
getprocinfo_te 2 22 13188
seqinc_prio     2 23 15724
seqdec_prio     2 24 15708
console         3 25 0
$ hello_test
helloxv6
$ helloname_test shlee
hello shlee
$ getnumproc_test
Total Number of Active Processes: 3
$ getmaxpid_test
Maximum PID: 7
$ getprocinfo_test
PID     PPID    SIZE    Number of Context Switch
1       0       12288   56
2       1       16384   74
8       2       12288   21
$
```

4 새로운 스케줄러 테스트 및 검증

seqinc_prio 테스트.

```
$ seqinc_prio 5
Priority of parent process = 1000

All children completed
Child 0.    pid 10
Child 1.    pid 11
Child 2.    pid 12
Child 3.    pid 13
Child 4.    pid 14

Exit order
pid 14
pid 13
pid 12
pid 11
pid 10
$ seqinc_prio 10
Priority of parent process = 1000

All children completed
Child 0.    pid 16
Child 1.    pid 17
Child 2.    pid 18
Child 3.    pid 19
Child 4.    pid 20
Child 5.    pid 21
Child 6.    pid 22
Child 7.    pid 23
Child 8.    pid 24
Child 9.    pid 25

Exit order
pid 25
pid 24
pid 23
pid 22
pid 21
pid 20
pid 19
pid 18
pid 17
pid 16
```

seqdec_prio 테스트

```
$ seqdec_prio 10
Priority of parent process = 1000

All children completed
Child 0.    pid 27
Child 1.    pid 28
Child 2.    pid 29
Child 3.    pid 30
Child 4.    pid 31
Child 5.    pid 32
Child 6.    pid 33
Child 7.    pid 34
Child 8.    pid 35
Child 9.    pid 36

Exit order
pid 27
pid 28
pid 29
pid 30
pid 31
pid 32
pid 33
pid 34
pid 35
pid 36
$ seqdec_prio 5
Priority of parent process = 1000

All children completed
Child 0.    pid 38
Child 1.    pid 39
Child 2.    pid 40
Child 3.    pid 41
Child 4.    pid 42

Exit order
pid 38
pid 39
pid 40
pid 41
pid 42
$
```

seqdec_prio, seqinc_prio 두 테스트 프로그램에서 모두 우선순위가 높은 프로세스부터 차례대로 수행, 종료됨을 확인하였다.

3. 소스코드

- **각 파일에서 추가 / 삭제한 코드**

defs.h

```
int                 get_num_proc(void);
int                 get_max_pid(void);
int                 get_proc_info(int, struct processInfo *);
int                 set_prio(int);
int                 get_prio(void);
```

getmaxpid_test.c

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


int main(int argc, char *argv[])
{
        printf(1, "Maximum PID: %d₩n", get_max_pid());
        exit();
}
```

getnumproc_test.c

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

int main(void)
{
        printf(1, "Total Number of Active Processes: %d₩n", get_num_proc());
        exit();
}
```

getprocinfo_test.c

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

int main(void)
{
        struct processInfo pInfo;
        int pid;
        int max_pid;
```

```
        printf(1, "PID₩tPPID₩tSIZE₩tNumber of Context Switch₩n");
        max_pid = get_max_pid();
        for (pid = 0; pid <= max_pid; ++pid) {
                if (get_proc_info(pid, &pInfo) != -1) {
                        printf(1, "%d₩t%d₩t%d₩t%d₩n", pid, pInfo.ppid, pInfo.psize, pInfo.numberContextSwitches);
                }
        }

        exit();
}
```

helloname_test.c

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

int main(int argc, char *argv[])
{
        if (argc < 2) {
                printf(2, "usage: <%s> name₩n", argv[0]);
                exit();
        }

        hello_name(argv[1]);
```

```
        exit();
}
```

proc.c

```
#include "processInfo.h"
#include "rand.h"

unsigned int get_ticket_total(void);
void update_ticket(struct proc *p);

@@ -112,6 +118,10 @@ allocproc(void)
  p->context_switch_count = 0; // get_proc_info를 위해 추가
  p->prio = 500; // prio에 올 수 있는 값의 중간값인 500으로 초기화


scheduler(void)
{
  struct proc *p;
  struct cpu *c = mycpu();
  c->proc = 0;
                    // 우선순위 RR 스케쥴링
                    for(;;){
                     // Enable interrupts on this processor.
                     sti();
```

```
// Loop over process table looking for process to run.
acquire(&ptable.lock);

max_prio = 0;
prio_total = 0;
runnable_count = 1;
for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
        if(p->state != RUNNABLE)
                continue;

        prio_total += p->prio;
        ++runnable_count;

        if(p == c->proc) // 실행중이었던 프로세스이면 count값을 변경하지 않는다
                continue;

        p->count += (MAX_COUNT - p->count) / 2; // 실행하지 않은 프로세스의 count값을 증가시킨다
        if (p->count > MAX_COUNT)
                p->count = MAX_COUNT;

        if (max_prio < p->prio + p->count) // 우선순위 최댓값을 기록한다
                max_prio = p->prio + p->count;
}

for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
        if(p->state != RUNNABLE)
```

```
                continue;

            if(max_prio <= p->prio + p->count) { // 실행해야 할 우선순위의 프로세스라면
                p->context_switch_count += 1; // context switch 횟수 + 1
                if(runnable_count == 0) runnable_count = 1;
                p->count -= (prio_total / runnable_count); // count값을 prio_total / runnable_count만큼
감소시킨다

                //cprintf("%d %d %d₩n", p->prio, p->count, p->prio + p->count);
                if (p->count <= 0) { // p->count가 0보다 작아졌다면 다시 MAX_COUNT로
                    p->count = MAX_COUNT;
                }
                // context swtiching
                c->proc = p;
                switchuvm(p);
                p->state = RUNNING;

                swtch(&(c->scheduler), p->context);
                switchkvm();

                // Process is done running for now.
                // It should have changed its p->state before coming back.
                c->proc = 0;
            }
        }

        release(&ptable.lock);
    }
```

```
}

int
get_num_proc(void)
{
        struct proc *p;
        int count = 0;
        acquire(&ptable.lock);

        for (p = ptable.proc; p < &ptable.proc[NPROC]; ++p) {
         if (p->state != UNUSED)
                ++count;
        }

        release(&ptable.lock);
        return count;
}

int
get_max_pid(void) {
        struct proc *p;
        int max_pid = 0;
        acquire(&ptable.lock);

        for (p = ptable.proc; p < &ptable.proc[NPROC]; ++p) {
         if (p->state != UNUSED && p->pid > max_pid)
                max_pid = p->pid;
```

```
        }

        release(&ptable.lock);
        return max_pid;
}

int
get_proc_info(int pid, struct processInfo *pInfo)
{
        struct proc *p;
        int process_found_flag = 0;
        acquire(&ptable.lock);

        for (p = ptable.proc; p < &ptable.proc[NPROC]; ++p) {
         if (p->state != UNUSED && p->pid == pid) {
                if (p->parent == 0) {
                        pInfo->ppid = 0;
                } else {
                        pInfo->ppid = p->parent->pid;
                }
                pInfo->psize = p->sz;
                pInfo->numberContextSwitches = p->context_switch_count;

                process_found_flag = 1;
                break;
         }
        }
```

```
                release(&ptable.lock);


                if (process_found_flag)
                 return 0;
                else
                 return -1;

}

int
set_prio(int n)
{
                // 우선순위는 양의 정수만 가능하다. 이 범위를 벗어나면 우선순위를 변경하지 않고 -1을 리턴한다
                if (0 < n) {
                 struct proc *p = myproc();
                 acquire(&ptable.lock);
                 p->prio = n;
                 release(&ptable.lock);

                 return 0;
                } else {
                 return -1;
                }
}

int
```

```
get_prio()
{
                int n = 0;

                acquire(&ptable.lock);
                n = myproc()->prio;
                release(&ptable.lock);

                // 우선순위는 양의 정수만 가능하다. 이 범위를 벗어나면 -1을 리턴한다
                if (0 < n) {
                 return n;
                } else {
                 return -1;
                }
}
```

proc.h

```
@@ -49,6 +49,9 @@ struct proc {
  struct file *ofile[NOFILE];   // Open files
  struct inode *cwd;            // Current directory
  char name[16];                // Process name (debugging)
  int context_switch_count;     // Number of context switches
  unsigned int prio;            // 스케쥴링 우선순위
  int count;                    // 우선순위 기반 RR 스케쥴링에 사용할 count};
}
```

syscall.c

```
extern int do_hello(void);
extern int do_hello_name(void);
extern int do_get_num_proc(void);
extern int do_get_max_pid(void);
extern int do_get_proc_info(void);
extern int do_set_prio(void);
extern int do_get_prio(void);

[SYS_hello]     do_hello,
[SYS_hello_name] do_hello_name,
[SYS_get_num_proc] do_get_num_proc,
[SYS_get_max_pid] do_get_max_pid,
[SYS_get_proc_info] do_get_proc_info,
[SYS_set_prio] do_set_prio,
[SYS_get_prio] do_get_prio,
```

syscall.h

```
#define SYS_hello   22
#define SYS_hello_name 23
#define SYS_get_num_proc 24
#define SYS_get_max_pid 25
#define SYS_get_proc_info 26
#define SYS_set_prio 27
#define SYS_get_prio 28
```

```
#define SYS_rand 28
#define SYS_srand 28
```

sysproc.c

```
#include "processInfo.h"

int
do_hello(void)
{
        cprintf("helloxv6₩n");

        return 0;
}

int
do_hello_name(void)
{
        char *name;
        argstr(0, &name);
        cprintf("hello %s₩n", name);

        return 0;
}

int
```

```
do_get_num_proc(void)
{
        return get_num_proc();
}

int
do_get_max_pid(void)
{
        return get_max_pid();
}

int
do_get_proc_info(void)
{
        int pid;
        struct processInfo *pInfo;

        argint(0, &pid);
        argptr(1, (void *)&pInfo, sizeof(pInfo));

        return get_proc_info(pid, pInfo);
}

int
do_set_prio()
{
        int n;
```

```
        argint(0, &n);
        return set_prio(n);
}

int
do_get_prio()
{
        return get_prio();
}
```

user.h

```
struct processInfo;

int hello(void);
int hello_name(const char *);
int get_num_proc(void);
int get_max_pid(void);
int get_proc_info(int, struct processInfo *);
int set_prio(int);
int get_prio(void);
//int rand(void);
//int srand(unsigned int);
```

usys.S

SYSCALL(hello)

SYSCALL(hello_name)

SYSCALL(get_num_proc)

SYSCALL(get_max_pid)

SYSCALL(get_proc_info)

SYSCALL(set_prio)

SYSCALL(get_prio)