소프트웨어학부 3학년 / 20180285 / 박민준

운영체제 3차과제

개발 환경 : vmware : ubuntu 20.04 :: Linux

수행 과정

0. XV6 운영체제의 스케줄러인 void scheduler(void) 함수 분석하기

```
struct proc *p, *tmp;
struct proc *bestproc;
struct cpu *c = mycpu();
c->proc = 0;
```

함수 내 사용할 프로세스 선언 뒤, cpu 초기화.

```
for(;;){
```

스케줄러는 무한한 루프 함수.

```
sti();
```

인터럽트가 가능하도록 반복문 내 함수 호출.

```
acquire(&ptable.lock);
for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
```

Ptable에 대한 lock을 얻은 후, ptable 내의 proc를 순회

```
if(p->state != RUNNABLE)
  continue;

// Switch to chosen process.  It is the process's job
// to release ptable.lock and then reacquire it
// before jumping back to us.
c->proc = p;
switchuvm(p);
p->state = RUNNING;
```

순회 중, p의 상태가 RUNNABLE 하다면 실행할 프로세스로 선택.

선택 후, cpu에 프로세스를 지정하고, switchuvm으로 상태를 변경한 뒤, 상태를 RUNNING으로 변경

이때, 계속해서 context switching 이 발생하며 RUNNABLE한 프로세스를 번갈아가며

실행하기 때문에 위 스케줄러는 라운드 로빈 스케줄러.

```
  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);
```

Context를 선정하였던 p의 context로 변경.

Kernel mode 로 변경 해준 뒤, cpu의 proc또한 초기화.

Ptable에 대한 lock을 반환.

위 과정을 반복.

1. 각 프로세스에 "priority" 개념 추가

- proc.h 파일안에 있는 "proc" 구조체에 "int priority" 멤버 추가

```
52    int priority;
53  };
```

멤버변수 priority 추가.

- 프로세스가 생성될 때, priority 값은 5 이어야 함 (최초로 실행되는 프로세스부터)

```
void
userinit(void)
{
  struct proc *p;
  extern char _binary_initcode_start[], _binary_initcode_size[];

  p = allocproc();
```

최초 프로세스를 생성하는 userinit 함수에서 p = allocproc(); 를 통해 프로세스르 할당.

```
static struct proc*
allocproc(void)
{
  struct proc *p;
  char *sp;

  acquire(&ptable.lock);

  for(p = ptable.proc; p < &ptable.proc[NPROC]; p++)
    if(p->state == UNUSED)
      goto found;

  release(&ptable.lock);
  return 0;

found:
  p->state = EMBRYO;
  p->pid = nextpid++;
  p->priority = 5;
```

allocproc 함수가 반환해주는 proc *p 변수에 p->priority = 5; 코드를 추가하여 priority 값을 5로 지정.

- fork()를 통해서 프로세스가 생성될 때 자식 process는 부모 process 와 같은 priority 값은 가져야함

```
int
fork(void)
{
  int i, pid;
  struct proc *np;
  struct proc *curproc = myproc();

  // Allocate process.
  if((np = allocproc()) == 0){
    return -1;
  }

  // Copy process state from proc.
  if((np->pgdir = copyuvm(curproc->pgdir, curproc->sz)) == 0){
    kfree(np->kstack);
    np->kstack = 0;
    np->state = UNUSED;
    return -1;
  }
  np->sz = curproc->sz;
  np->parent = curproc;
  *np->tf = *curproc->tf;
  np->priority = curproc->priority;
```

fork 함수 에서는 자식 프로세스인 proc *np 변수에 np의 priority는 curproc의 값과 같게 해주는 코드 추가

np ->priority = curproc->priority;

- "set_proc_priority" 시스템 콜 구현

```
void sys_set_proc_priority(int pid, int priority){
  struct proc *p;
  int pid_sys;
  int priority_sys;
  argint(0,&pid_sys);
  argint(1,&priority_sys);
  acquire(&ptable.lock);

  for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
    if(p->pid == pid_sys){
      p->priority = priority_sys;
      break;
    }
  }
  release(&ptable.lock);
  return;
}
```

- "get_proc_priority" 시스템 콜 구현

```
int sys_get_proc_priority(int pid){
  struct proc *p;
  int pid_sys;
  argint(0,&pid_sys);

  acquire(&ptable.lock);
  int priority = -1;
  for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
    if(p->pid == pid_sys){
      priority = p->priority;
      break;
    }
  }
  release(&ptable.lock);
  return priority;
}
```

시스템 콜 등록 과정

<syscall.h>

```
24  #define SYS_set_proc_priority   23
25  #define SYS_get_proc_priority   24
```

<syscall.c>

```
107   extern int sys_set_proc_priority(void);
108   extern int sys_get_proc_priority(void);
```

```
134   [SYS_set_proc_priority] sys_set_proc_priority,
135   [SYS_get_proc_priority] sys_get_proc_priority,
```

<user.h>

```
27  int set_proc_priority(int pid, int priority);
28  int get_proc_priority(int pid);
```

<usys.S>

```
33    SYSCALL(set_proc_priority)
34    SYSCALL(get_proc_priority)
```

2. void scheduler(void) 함수 수정하여 Priority Scheduler 구현

```
struct proc *bestproc;
```

bestproc 추가

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

if (bestproc != 0 && bestproc->state == RUNNABLE) {
  p = bestproc;
}
```

실행할 프로세스를 priority 비교를 통해 선정.

tmp의 priority 가 더 작다면 해당 프로세스를 bestproc로 지정.

마지막으로 bestproc를 현재 실행할 p프로세스로 설정.

<테스트 시나리오>

0. 현재 프로세스의 상태를 확인할 수 있는 state시스템콜 추가.

```
int state(void){
  struct proc *p;

  sti();
  acquire(&ptable.lock);

  cprintf("name \t\t pid \t state \t priority \t cnt \n");
  for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
    if(p->state == SLEEPING){
      cprintf("%s \t\t %d \t SLEEPING \t %d \t %d \n", p->name,p->pid,p->priority,p->cnt);
    }else if(p->state == RUNNING){
      cprintf("%s \t\t %d \t RUNNING \t %d \t %d \n ", p->name,p->pid,p->priority,p->cnt);
    }else if(p->state == RUNNABLE){
      cprintf("%s \t\t %d \t RUNNABLE \t %d \t %d \n ", p->name,p->pid,p->priority,p->cnt);
    }
  }
  release(&ptable.lock);

  return 0;
}
```

결과화면

```
$ state
name            pid     state   priority        cnt
init            1       SLEEPING        5       13
sh              2       SLEEPING        5       17
state           3       RUNNING         5       7
 $
```

1. Proc 구조체에 실행된 횟수를 저장하는 변수 cnt 추가.

```
  int cnt;
```
proc.h

```
np->cnt = 0;
```
fork에서 cnt초기화

```
p->cnt = 0;
```
allocproc에서 cnt초기화

```
c->proc = p;
p->cnt++;
switchuvm(p);
p->state = RUNNING;
swtch(&(c->scheduler), p->context);
switchkvm();
```

cnt는 scheduler함수 내에서 해당 프로세스로 context switching이 일어날 경우 1씩 증가함.

2. 더미 프로세스 생성

   nproc 명령어 구현 – 프로세스 런타임을 늘리기 위해 다음과 같은 for문 구성.

```c
int
main(int argc, char *argv[])
{
  double x = 0.00;
  for(float j=0.1;j<100000.0;j+=0.001){
    x += 11.11 * 11.11;
  }

  exit();
}
```

nproc &; nproc &;

3. State 시스템 콜을 통해 현재 모든 프로세스의 상태를 확인

State

```
$ nproc &;
$ nproc &;
$ state
name            pid     state   priority        cnt
init            1       SLEEPING        5       19
sh              2       SLEEPING        5       33
state           9       RUNNING         5       8
 nproc          5       RUNNABLE        5       590
 nproc          8       RUNNING         5       404
 $
```

4. 실행중인 더미 프로세스 하나의 priority를 7로 변경 (set_proc_priority를 활용한 chpr.c 사용)

```c
int
main(int argc, char *argv[])
{
  int priority, pid;
  pid = atoi(argv[1]);
  priority = atoi(argv[2]);
  if (priority < 0 || priority > 10){
    printf(2,"Priority should 0-10!\n");
    exit();
  }
  set_proc_priority(pid, priority);
  exit();
}
```

chpr 5 7

5. 새로운 더미 프로세스 생성 (priority는 default 5)

   nproc &;

6. State 시스템 콜을 통해 priority가 변경된 프로세스의 상태 확인

   State

```
$ chpr 5 7
$ nproc &;
```

```
$ state
name            pid     state   priority        cnt
init            1       SLEEPING        5       19
sh              2       SLEEPING        5       44
state           14      RUNNING         5       2
 nproc          5       RUNNABLE        7       3036
 nproc          8       RUNNABLE        5       3026
 nproc          13      RUNNING         5       179
$
```

----------proc의 cnt 변수를 통해 선점형 방식 스케줄러 확인----------

7. Priority가 7인 프로세스는 Starvation 상태가 됨. Priority 가 7인 프로세스의 cnt 값이 변동되지 않음을 확인.

   라운드로빈 스케줄러에서 실행 횟수를 저장하는 cnt 변수가 증가하지 않고 그대로인 모습이 관측됨.

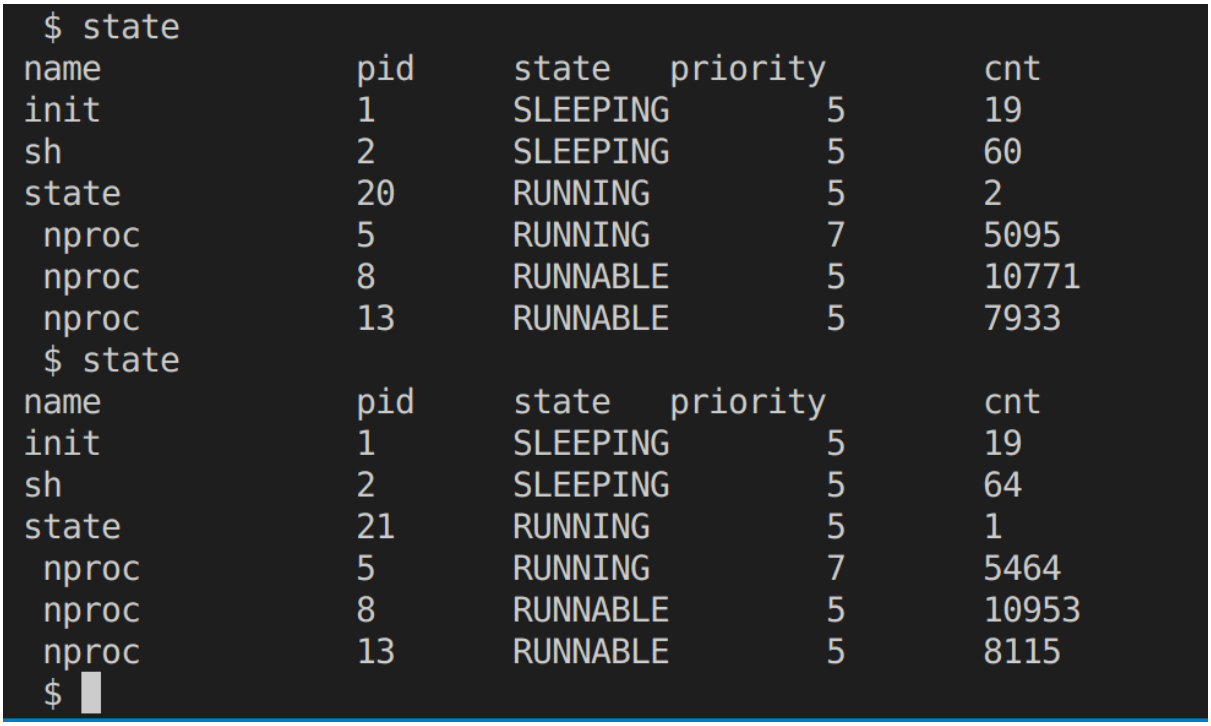
```
 $ state
name            pid     state   priority        cnt
init            1       SLEEPING        5       19
sh              2       SLEEPING        5       46
state           15      RUNNING         5       1
 nproc          5       RUNNABLE        7       3036
 nproc          8       RUNNABLE        5       4970
 nproc          13      RUNNING         5       2127
 $
```

----------starvation 현상이 나타나는 모습을 직접 관측----------

8. bestproc를 지정할 때 priority 우선순위가 낮고(값이 더 크고), 실행횟수(cnt)가 5000이상 차이 날 경우, priority와 관계없이 bestproc로 지정되도록 코드 작성.

```
}else if(bestproc->priority < tmp->priority && bestproc->cnt - tmp->priority > 5000){
  bestproc = tmp;
  break;
}
```

9. 실행 결과.



두번의 state 결과 priority가 7인 프로세스가 starvation 현상이 생기지 않고 cnt가 증가함을 확인할 수 있음.

----------실행횟수를 기억하여 starvation 현상 자체 해결----------

발생한 문제점.

1. Starvation 관측 과정에서 많은 어려움을 겪음.

- Xv6의 경우 프로세스 실행 시 nproc와 같이 해주면 명령어 실행 도중 다른 명령어를 사용하기 어려움. 이에 nproc &; 같은 &인자를 더해주어 명령어 실행 중에도 다른 명령어를 사용할 수 있게 해 줌.

2. Set, get proc_priority 구현 중 오류

- 시스템콜로 user program 에서 입력받은 pid를 전달 하는 방법을 지난 과제를 참고하여 해결. argint 함수 사용