컴퓨터학부 20201853 서민비

1. 개요

프로세스마다 가중치와 우선순위를 부과하여 기존의 RR방식(process table을 순회하며 RUNNALBLE 상태인 프로세스를 순차적으로 선택하는 방식)이었던 xv6 스케줄링 기법 대신 높은 우선순위를 가진 프로세스를 선택하여 실행하는 ssu scheduler를 구현한다. 프로세스의 weight는 1부터 차례대로 증가시키며 부여하지만, sdebug 명령어에 의해 생성되는 프로세스의 weight값은 시스템에서 생성한 프로세스와 별도로 생성된 순서에 따라 1부터 증가시키며 weight값을 부여한다. p프로세스의 default priority값은 init process가 처음 생기는 시점에서 값을 3으로 지정하고, 스케줄링 함수가 호출될 때마다 갱신된다. 가장 작은 priority값은 ptable의 멤버로 관리하며 역시 스케줄링 함수가 호출될 때마다 priority와 함께 갱신된다. 단, 프로세스 생성 또는 wake up시 프로세스의 priority 값 중 가장 작은 값을 부여한다. 구현한 스케줄링 함수는 RUNNABLE인 프로세스들을 순회하면서 가장 작은 priority 값을 프로세스를 선택해 실행하는데 이러한 스케줄링 함수의 동작 과정을 확인하기 위해 프로세스 정보(프로세스 id, 프로세스 가중치, 프로세스 생성부터 프로세스 정보 출력까지 소요 시간)을 출력하는 sdebug 명령어도 함께 구현한다.

2. 결과(실행결과 및 설명)

-1번, 2번

텍스트이(가) 표시된 사진

자동 생성된 설명

첫 번째 sdebug 입력 결과

텍스트, 스크린샷, 검은색, 전자기기이(가) 표시된 사진

자동 생성된 설명

두 번째 sdebug 입력 결과

텍스트, 스크린샷, 모니터, 검은색이(가) 표시된 사진

자동 생성된 설명

세 번째 sdebug 입력 결과

\*첫 번째 sdebug에서 initcode, intit, sh 다음으로 fork()를 PNUM만큼 수행함으로써 pid=4,5,6,7,8을 가지는 프로세스가 생성되었다가 우선순위가 높은 pid=8,7,6,5,4 순으로 종료된 것을 확인할 수 있다. 두 번째 sdebug에서도 PNUM에 따라 fork()를 통해 pid=10,11,12,13,14을 가지는 프로세스가 생성되었다가 우선순위가 높은 pid=14,13,12,11,10 순으로 종료된 것을 확인할 수 있으며, 세 번째 sdebug에서도 같은 원리로 위와 같은 결과가 출력된다.

-3번 : 구현하지 못했습니다. 죄송합니다.

3. 소스코드

Makefile

ifndef CPUS

CPUS := 1 //cpu core=1

UPROGS=\

    \_cat\

    \_echo\

    \_forktest\

    \_grep\

    \_init\

    \_kill\

    \_ln\

    \_ls\

    \_mkdir\

    \_rm\

    \_sh\

    \_stressfs\

    \_usertests\

    \_wc\

    \_zombie\

    \_sdebug\

...

EXTRA=\

    mkfs.c ulib.c user.h cat.c echo.c forktest.c grep.c kill.c\

    ln.c ls.c mkdir.c rm.c stressfs.c usertests.c wc.c zombie.c\

    printf.c umalloc.c sdebug.c\

    README list.txt dot-bochsrc \*.pl toc.\* runoff runoff1 runoff.list\

    .gdbinit.tmpl gdbutil\

usys.S

SYSCALL(weightset)

syscall.h

#define SYS\_weightset 22

syscall.c

extern int sys\_weightset(void);

static int (\*syscalls[])(void) = {

...

[SYS\_weightset]   sys\_weightset,

};

user.h

int weightset(int);

sysproc.c

int

sys\_weightset(void)

{

  //매개변수로 입력받은 값을 sdebug 명령어에 의해 생성되는 프로세스의 weight값으로 부여하도록 구현

  int weight;

  if (argint(0, &weight) < 0)

    return -1;

  // if(weight==0)

  //   return -1;

  // acquire(&ptable.lock);

  // struct proc \*p=myproc();

  // p->weight=weight;  //set weight

  do\_weightset(weight);

  //cprintf("sys\_weightset%dpid %dweight and %dpriority\n",p->pid,p->weight,p->priority);

  // release(&ptable.lock);

  return 0;

}

sdebug.c

#include "types.h"

#include "stat.h"

#include "user.h"

#include "fcntl.h"

#include "syscall.h"

#include "param.h"

#define PNUM 5  //proess 개수

#define PRINT\_CYCLE 10000000   //process 정보 출력 주기

#define TOTAL\_COUNTER 500000000 //프로세스 종료되기까지 시간

void sdebug\_func(void)

{

    //implement

    //int sdebug\_weight=1;

    printf(1,"start sdebug command\n");

    for(int i=0;i<PNUM;i++){

        int pid;

        pid = fork(); //스레드생성. 성공 시 자식 프로세스 생성 후 부모 프로세서와 함께 돌아감.

        if(pid < 0){

            printf(1, "sdebug: fork failed\n");

            exit();

        }else if(pid==0){

            //자식프로세스인 경우

            //weightset(), scheduler()호출

            int starttik,endtick;

            starttik=uptime();

            weightset(i+1);

            long counter=0;

            while(1){

                counter++;

                if(counter==PRINT\_CYCLE){

                    endtick=uptime();

                    printf(1,"PID : %d WEIGHT : %d TIMES : %d ms\n",getpid(),i+1,((endtick-starttik)\*10));

                }

                if(counter>TOTAL\_COUNTER){

                    break;

                }

            }

            printf(1, "PID : %d is terminated\n",getpid());

            exit();

        }

    }

    for(int n=0; n <PNUM; n++){

        wait();

    }

    printf(1,"end of sdebug command\n");

    return;

}

int main(int argc, char \*\*argv)

{

    sdebug\_func();

    exit();

}

proc.h

struct proc {

  uint sz;                     // Size of process memory (bytes)

  pde\_t\* pgdir;                // Page table

  char \*kstack;                // Bottom of kernel stack for this process

  enum procstate state;        // Process state

  int pid;                     // Process ID

  struct proc \*parent;         // Parent process

  struct trapframe \*tf;        // Trap frame for current syscall

  struct context \*context;     // swtch() here to run process

  void \*chan;                  // If non-zero, sleeping on chan

  int killed;                  // If non-zero, have been killed

  struct file \*ofile[NOFILE];  // Open files

  struct inode \*cwd;           // Current directory

  char name[16];               // Process name (debugging)

  long priority; //우선순위

  int weight;  //가중치

};

defs.h

void            do\_weightset(int weight);

void            update\_priority(struct proc \*proc);

void            update\_min\_priority();

void            assign\_min\_priority(struct proc \*proc);

proc.c

#define TIME\_SLICE 10000000

#define NULL ((void \*)0)

struct {

  struct spinlock lock;

  struct proc proc[NPROC];

  long minpriority;

} ptable;

int nextpid = 1;

int nextweight=1;

void update\_priority(struct proc \*proc)

{

  proc->priority=proc->priority+(TIME\_SLICE/proc->weight);

  //cprintf("do\_weightset %dpid %dweight and %dpriority\n",proc->pid,proc->weight,proc->priority);

}

void update\_min\_priority()

{

  //struct proc \*min=0;

  //struct proc \*p;

  struct proc \*p1=0;

  struct proc \*minP=0;

    minP =ptable.proc;

    //cprintf("minP is %dpid\n", minP->pid);

    for(p1 = ptable.proc; p1 < &ptable.proc[NPROC]; p1++){

      if((p1->state == RUNNABLE) && (minP->priority > p1->priority))

        minP = p1;

    }

    if(minP != 0)

      ptable.minpriority=minP->priority;

  //cprintf("minP is %d in %dpid\n",ptable.minpriority, minP->pid);

}

void assign\_min\_priority(struct proc \*proc){

  proc->priority=ptable.minpriority;

}

static struct proc\*

allocproc(void)

{

...

found:

  p->state = EMBRYO;

  p->pid = nextpid++;

  p->weight=nextweight++; //default weight

  assign\_min\_priority(p);

release(&ptable.lock);

...

}

void

userinit(void)

{

  struct proc \*p;

  extern char \_binary\_initcode\_start[], \_binary\_initcode\_size[];

  ptable.minpriority=3;

p = allocproc();

...

}

void

scheduler(void)

{

  struct proc \*p = 0;

  struct cpu \*c = mycpu();

  c->proc = 0;

  for(;;)

  {

    sti();

    acquire(&ptable.lock);

    for(p = ptable.proc; p < &ptable.proc[NPROC]; p++)

    {

      struct proc \*highP = 0;

      struct proc \*p1 = 0;

      if(p->state != RUNNABLE)

        continue;

      highP = p;

      for(p1 = ptable.proc; p1 < &ptable.proc[NPROC]; p1++){

        if((p1->state == RUNNABLE) && (highP->priority > p1->priority))

          highP = p1;

      }

      if(highP != 0)

        p = highP;

      if(p != 0)

      {

        c->proc = p;

        switchuvm(p);

        p->state = RUNNING;

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

        switchkvm();

        update\_priority(p);

        update\_min\_priority();

        //cprintf("sched() %dpid %dweight %dpriority\n",p->pid,p->weight,p->priority);

        c->proc = 0;

      }

    }

    release(&ptable.lock);

  }

}

static void

wakeup1(void \*chan)

{

  struct proc \*p;

  for(p = ptable.proc; p < &ptable.proc[NPROC]; p++)

    if(p->state == SLEEPING && p->chan == chan){

      assign\_min\_priority(p);

      p->state = RUNNABLE;

    }

}

void do\_weightset(int weight)

{

  acquire(&ptable.lock);

  struct proc \*p=myproc();

  p->weight=weight;  //set weight

  release(&ptable.lock);

  //cprintf("do\_weightset() %dpid %dweight and %dpriority\n",p->pid,p->weight,p->priority);

}