به نام خدا

دانشکده مهندسی کامپیوتر

آزمایشگاه سیستم های عامل

آزمایش هشتم : زمانبندیCPU

اعضای گروه :

محمد امین فرح بخش - (40131029)

حسین تاتار – (40133014)

اردیبهشت 1404

الگوریتم زمانبندی CFS :

در این آزمایش میخواهیم یک نمونه ساده شده از الگوریتم زمانبندی CFS را پیاده سازی کنیم. ابتدا پروژه را از این لینک clone میکنیم ، و سپس توابع فایل cfs_scheduler. c را پیاده سازی میکنیم :

: Fill_process_array

این تابع باید به تعداد PROCESS_COUNT پردازه ساخته و آرایه پردازه ها را با آنها پر کند. پس با یک حلقه for و به کمک تابع create_process ، هر پردازه را با شناسه ، vruntime و مدت زمان باقیمانده مشخص ساخته و داخل اندیس آن در آرایه ذخیره میکنیم.

```
9   void fill_process_array(process *process_array[PROCESS_COUNT])
10  {
11      // fill the process_array with processes
12      for (int i = 0; i < PROCESS_COUNT; i++)
13      {
14          process_array[i] = create_process(2000 + i, // process id
15                                            100 + i,  // current vruntime
16                                            3);       // residual execution duration of the process
17      }
18  }
19
```

: Insert_one_process_to_rbtree

این تابع باید با گرفتن اشاره گر درخت سیاه-قرمز و پردازه موردنظر، آن راداخل درخت ذخیره کند. برای اینکار ابتدا یک نمونه از نوعmydata با کلید proc->vruntime برای آن پردازه ساخته و این نمونه را به کمک rb_insert در درخت ذخیره کنیم. در صورت ایجاد خطا در ذخیره پردازه پیام مناسب چاپ میشود.

```
19
20  void insert_one_process_to_rbtree(rbtree *rbt, process *proc)
21  {
22      // make mydata instance of process and insert it to the rbtree, the key is proc->vruntime
23      mydata *data = makedata_with_object(proc->vruntime, proc);
24      if (rb_insert(rbt, data) == NULL)
25      {
26          fprintf(stderr, "insert %d: out of memory\n", proc->id);
27      }
28  }
29
```

: Insert_processes_to_rbtree

این تابع باید با گرفتن اشاره گر درخت قرمز-سیاه و آرایه پردازه ها ، تمامی اعضای آرایه را در درخت ذخیره کند. پس کافیست به کمک تابع قبل و یک حلقه for، با پیمایش آرایه به کمک proc ،اعضای آرایه را در درخت ذخیره کنیم.

```
void insert_processes_to_rbtree(rbtree *rbt, process *process_array[PROCESS_COUNT])
{
    // insert processes in process_array to rbtree
    process *proc;
    for (int i = 0; i < PROCESS_COUNT; i++)
    {
        proc = process_array[i];
        insert_one_process_to_rbtree(rbt, proc);
    }
}
```

: Process_of_node

این تابع با گرفتن یک گره از درخت، باید پردازه ذخیره شده در آن رابرگرداند. برای این کار کافیست مقدار data در استراکتrbnode رابه اشاره گری از نوع mydata تبدیل کرده و در آن استراکت، مقدار object را در اشاره گر پردازه ذخیره کرده وآن را برگردانیم.

```
process *process_of_node(rbnode *node)
{
    // extract the process in node
    process *proc = (process *)((mydata *)(node->data))->object;
    return proc;
}
```

: Main

ابتدا یک آرایه ساخته و به کمک تابع fill_process_array آن را از پردازه ها پر میکنیم.

سپس یک درخت قرمز-سیاه ساخته و در صورت خطا پیام مناسب چاپ میکنیم .

سپس پردازه های ساخته شده را به کمک insert_processes_to_rbtree ، پردازه ها را در درخت ذخیره میکنیم.

اکنون تنها کافیست الگوریتم زمانبندی را با یک حلقه پیاده سازی کنیم، برای این کار ابتدا متغیرهایی که درون حلقه به آنها نیاز داریم را تعریف کنیم : این متغیر ها شامل : گره فعلی درخت (node) ، پردازه در حال اجرا (current_proc) و تیک فعلی (current_tick) است.

برای قسمت اصلی الگوریتم ، باید در هر تکرار حلقه ، چپ ترین گره درخت انتخاب شود (به کمک RB_MINIMAL) ، پردازه داخل این گره به اندازه یک تیک اجرا شده و در صورت اتمام زمان باقی مانده آن از درخت حذف میشود .

در این حلقه ، ابتدا پردازه با کمترین vruntime (چپ ترین گره) انتخاب شده، به کمکprocess_of_node، پردازه داخل آن را بدست آورده ، و به کمک run_process_for_one_tick در فایل proc.h پروژه به اندازه یک تیک اجرا شده و سپس از درخت حذف میشود. در صورت تمام نشدن پردازه (یعنی صفر نبودن residual_duration که به کمک is_terminated بررسی میشود)دوباره پردازه را با مقدار vruntime جدید به درخت اضافه میکنیم .

```c
int main()
{
    // create an array of procdesses and fill it
    process *processes[PROCESS_COUNT];
    fill_process_array(processes);
    printf("process array filled\n");
    printf("process array filled\n");
    // create a red-black tree
    rbtree *rbt;
    if ((rbt = rb_create(compare_func, destroy_func)) == NULL)
    {
        fprintf(stderr, "create red-black tree failed\n");
    }
    else
    {
        printf("tree created\n");
    }
    // insert processes to rbtree
    insert_processes_to_rbtree(rbt, processes);
    printf("inserted processes to rbtree\n");

    // declare node, current_proc and current_tick
    rbnode *node;
    process *current_proc;
    int current_tick = 0;

    // scheduling algorithm:
    while ((node = RB_MINIMAL(rbt)))
    {
        printf("current_tick: %d\n", current_tick);
        current_tick++;
        current_proc = process_of_node(node);
        run_process_for_one_tick(current_proc);
        rb_delete(rbt, node, 0);
        if (!is_terminated(current_proc))
        {
            insert_one_process_to_rbtree(rbt, current_proc);
        }
    }
}
```

خروجی برنامه :

```
amin@Frb:~/completely-fair-scheduler$ ./cfs.sh
process array filled
process array filled
tree created
inserted processes to rbtree
current_tick: 0
process 2000 is running current vruntime: 100   current residual_duration: 3
after running for one tick:     vruntime: 101   residual_duration: 2
current_tick: 1
process 2001 is running current vruntime: 101   current residual_duration: 3
after running for one tick:     vruntime: 102   residual_duration: 2
current_tick: 2
process 2000 is running current vruntime: 101   current residual_duration: 2
after running for one tick:     vruntime: 102   residual_duration: 1
current_tick: 3
process 2002 is running current vruntime: 102   current residual_duration: 3
after running for one tick:     vruntime: 103   residual_duration: 2
current_tick: 4
process 2001 is running current vruntime: 102   current residual_duration: 2
after running for one tick:     vruntime: 103   residual_duration: 1
current_tick: 5
process 2000 is running current vruntime: 102   current residual_duration: 1
after running for one tick:     vruntime: 103   residual_duration: 0
process 2000 terminated.
current_tick: 6
process 2002 is running current vruntime: 103   current residual_duration: 2
after running for one tick:     vruntime: 104   residual_duration: 1
current_tick: 7
process 2001 is running current vruntime: 103   current residual_duration: 1
after running for one tick:     vruntime: 104   residual_duration: 0
process 2001 terminated.
current_tick: 8
process 2002 is running current vruntime: 104   current residual_duration: 1
after running for one tick:     vruntime: 105   residual_duration: 0
process 2002 terminated.

real    0m0.002s
user    0m0.000s
sys     0m0.001s
amin@Frb:~/completely-fair-scheduler$
```

در هر تیک، اطلاعات زیر نمایش داده میشود :

Current_tick یا شماره تیک جاری.

پردازه ای که در حال اجرا است و مقادیر vruntime و residual_duration آن قبل وبعد اجرا .

در صورتی که پردازه مدت زمان باقیمانده صفر داشته باشد، پیام اتمام آن چاپ میشود .

در نهایت نیز زمان اجرای real ، user و sys را نشان میدهد .

ترتیب اجرای پردازه ها :

در ابتدا با توجه به تابع ایجاد پردازه ها ،در آرایه پردازه ها داریم :

- Index 0 : id = 2000 , vruntime = 100 , residual_duration = 3
- Index 1: id = 2001 , vruntime = 101 , residual_duration = 3
- Index 2: id = 2002 , vruntime = 102 , residual_duration = 3

درهر تیک ، پردازه با مقدار vruntime مینیمم انتخاب شده ، اجرا می شود و مقدار vruntime آن افزایش پیدا میکند . پس ابتدا پردازه 2000 انتخاب شده و vruntime آن 101 میشود.

سپس پردازه 2001 و مقدار vruntime آن102 میشود؛ سپس مینیمم vruntime ،101 است مربوط به پردازه 2000 پس آن اجرا میشود و vruntime آن102 میشود و سپس پردازه 2002 شانس اجرا را پیدا میکند.

و به همین ترتیب در هر گام پردازه با vruntime مینیمم انتخاب شده و اجرا می شود و مقدار vruntime آن تغییر کرده وسپس دوباره پردازه بعدی انتخاب شده و...